Towards Large-scale Network Analytics

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Xintian Yang, M.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2012

Dissertation Committee:

Srinivasan Parthasarathy, Advisor
P. Sadayappan
Gagan Agrawal
Abstract

In this thesis, we present a framework for efficient analysis of large-scale network datasets. There are four important components in our framework: a) a high performance computing platform with Graphics Processing Units (GPUs) and efficient implementations of mining algorithms on top of the GPU platform. b) an efficient summarization method to compress the storage space of large-scale streaming and heterogeneous network data with textual content and network topology. c) a complex query engine that depends on the summarized input data and can help to discover new knowledge from network content and topology. d) a visual front-end to present mining results to users.

First, the key challenge we address in this work is that of scalability – handling large datasets in terms of the efficiency of the back-end mining algorithms. Several of the mining algorithms that we have investigated share a common Sparse Matrix-Vector Multiplication (SpMV) kernel. We present a novel approach to compute this kernel on the GPUs, particularly targeting sparse matrices representing graphs with power-law attributes. Using real web graph data, we show how our representation scheme, coupled with a novel tiling algorithm, can yield significant benefits over the current state of the art GPU efforts on a number of core data mining algorithms such as PageRank, HITS and Random Walk with Restart. We also extend this efficient single GPU kernel to a cluster environment with multiple GPUs. The multi-GPU
kernel enables our framework to handle out-of-core datasets such as the Web graph. Additionally, the high performance of GPU kernel relies on programmer expertise and careful tuning of the parameters. We proposed an online parameter auto-tuning method with offline benchmarking component to accurately predict the parameters depending on the input data characteristics.

Second, we proposed an efficient summarization method to build an in-memory summary of high speed streaming network input data, which can contain both user generated content and topological information about user connections. The summary can be used as input to our framework to perform analytical tasks. Experimental results show that our method can efficiently and incrementally summarize the stream data. The memory footprint of the summarization algorithm grows logarithmically with time instead of linearly. The raw data can be approximately reconstructed by querying the summary so as to support the analytical applications over the original data.

Third, we proposed new complex queries on the summarized network content and topology. The queries about network content not only can detect popular topics discussed by users within the network during a period of time, but also can capture the evolution of such topics over time. The queries about the network topology project the entire network topology onto a subgraph conditioned on a network content keyword and a time interval. Graph mining queries are performed on such subgraphs to find relevant users of a topic keyword, communities of users for the keyword, or dynamic community events among a set of users.

Finally, we developed a visual-analytic toolkit for the interrogation of graph data such as those found in social, bibliometric, WWW and biological applications. The
tool we have developed incorporates common visualization paradigms such as zooming, coarsening and filtering while naturally integrating information extracted by data mining algorithms. The visual front-end provides features that are specifically useful in the analysis of graphs, capturing the static and dynamic nature of both individual entities as well as interactions among them. The tool provides the user with the option of selecting multiple views, designed to capture different aspects of the underlying graph data from the perspective of a node, a community or a subset of nodes of interest. Standard visual templates and cues are used to highlight critical changes that have occurred in dynamic graphs. Two case studies based on bibliometric and Wikipedia data are presented to demonstrate the utility of the toolkit for visual knowledge discovery.

In each of the above components, we propose new methods to either speed up mining tasks or reduce the data storage size in those tasks. We compare our methods with existing approaches on real datasets drawn from various domains.
To my family.
Acknowledgments

First of all, I sincerely thank my advisor Dr. Srinivasan Parthasarathy, for his patience and guidance in my PhD study. Without his support, I could not make it to the end of this journey.

I would also like to thank Dr. Gagan Agrawal and Dr. P. (Saday) Sadayappan for serving on my candidacy and dissertation committee. I am grateful for Dr. Sadayappan’s insightful advice in the GPU work we published together.

Former and present members of the Data Mining Research Lab have been very supportive and helpful both inside and outside school. They are: Matthew Otey, Sameep Mehta, Keith Marsolo, Amol Ghoting, Chao Wang, Gregory Buehrer, Sitaram Asur, Duygu Ucar, Shirish Tatikonda, Venu Satuluri, Matt Goyder, Ye, Wang, S.M. Faisal, Tyler Clemons, Yu-keng Shih, Yiye Ruan, Dave Fuhr and Yang Zhang. I would like to thank them along with my advisor, for the research discussions, project and paper collaborations. presentation rehearsals, paper proof readings and many other things.

I would like to give my special thanks to Sitaram Asur, Amol Ghoting and Matt Otey. Sitaram gave me a lot of help on my first paper submission. Amol Ghoting mentored me during my internship in IBM research. Matt Otey was instrumental in referring me to the summer internship position in Google which eventually led to my full time job.
I am grateful to the National Science Foundation for supporting my research through grants RI-CNS-0403342, CCF-0702587, CAREER-IIS-0347662, IIS-0917070 and SoCS-1111118. Any opinions, findings, and conclusions or recommendations expressed here are those of the author and, if applicable, his adviser and collaborators, and do not necessarily reflect the views of the National Science Foundation.

Finally and the most importantly, I would like to thank my wife, my mom and dad, and my parents in-law. In the long journey of pursuing my PhD degree, various good and bad things can happen and make the final goals crumbling. The unconditional supports from my family can always strengthen my determination to conquer all the difficulties.
**Vita**

November 27, 1983 ....................... Born - Harbin, China

July, 2006 ............................. B.E. Computer Science and Technology, Harbin Institute of Technology, China

Sept, 2004 - Jan, 2005 ................. Exchange Student, University of Hong Kong, Hong Kong

June, 2010 - Sept, 2010 ................. Research Intern, IBM T.J. Watson Research Center, Yorktown Heights, NY

Nov, 2010 ............................... M.S. Computer Science and Engineering, Ohio State University, Columbus, OH

June, 2011 - Sept, 2011 ................. Software Engineering Intern, Google, Kirkland, WA

Sept, 2010 - June, 2012 ................. Graduate Teaching Associate, Ohio State University, Columbus, OH

Sept, 2007 - present ..................... Graduate Research Associate, Ohio State University, Columbus, OH

**Publications**

**Research Publications**


Fields of Study

Major Field: Computer Science and Engineering
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Dedication</td>
<td>v</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>vi</td>
</tr>
<tr>
<td>Vita</td>
<td>viii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xiii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xv</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Challenges in Analyzing Large Graphs</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Proposed Framework</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Our Contributions</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Organization</td>
<td>10</td>
</tr>
<tr>
<td>2. Background and Related Work</td>
<td>11</td>
</tr>
<tr>
<td>2.1 Graph Mining Algorithms</td>
<td>11</td>
</tr>
<tr>
<td>2.1.1 Link Analysis</td>
<td>11</td>
</tr>
<tr>
<td>2.1.2 Graph Clustering</td>
<td>16</td>
</tr>
<tr>
<td>2.1.3 Dynamic Graph Analysis</td>
<td>20</td>
</tr>
<tr>
<td>2.2 GPU Background</td>
<td>26</td>
</tr>
<tr>
<td>2.3 Sparse Matrix and Vector Multiplication</td>
<td>28</td>
</tr>
<tr>
<td>2.4 Summarization of network content and topology</td>
<td>33</td>
</tr>
<tr>
<td>2.5 Visual Analytics</td>
<td>34</td>
</tr>
</tbody>
</table>
3. High Performance Mining Kernels ........................................ 36
   3.1 Methodology ....................................................... 39
      3.1.1 Single-GPU SpMV .............................................. 39
      3.1.2 Multi-GPU SpMV ............................................. 47
      3.1.3 Automatic Parameter Tuning .................................. 49
   3.2 Experiments ....................................................... 55
      3.2.1 Dataset and Hardware Detail .................................. 56
      3.2.2 Single-GPU SpMV Kernel .................................... 57
      3.2.3 Graph Mining Applications ................................... 62
      3.2.4 Multi-GPU PageRank on Web Graphs ......................... 66
      3.2.5 Parameter Auto-tuning ..................................... 67
   3.3 Discussion ....................................................... 69
   3.4 Conclusions ..................................................... 72

4. Compact Storage of Social Network Content and Topology ............ 74
   4.1 Introduction ..................................................... 74
   4.2 Stream Summarization of Network Content .......................... 77
      4.2.1 SPUR .......................................................... 78
      4.2.2 D-SPUR ...................................................... 86
   4.3 Compression of Network Topology with G-SPUR ....................... 91
   4.4 Experimental Results ............................................ 94
      4.4.1 Dataset and Setup .......................................... 95
      4.4.2 Batch Compression with SPUR ................................. 95
      4.4.3 Stream Summarization with D-SPUR ......................... 99
      4.4.4 Speeding up Graph Mining with G-SPUR on CPU and GPU .... 104
   4.5 Conclusions ................................................... 107

5. Complex Querying and Mining of Social Network Content and Topology 109
   5.1 Topic Event Detection Queries ................................... 110
      5.1.1 Live Analytics with TED-SPUR ............................... 110
      5.1.2 Analytical Query Results with TED-SPUR .................. 112
   5.2 Content and Time Aware Network Topology Queries ................ 116
      5.2.1 Content and Time Aware Subgraph Construction ............. 117
      5.2.2 Mining Algorithms on Subgraphs of Network Topology ....... 121
   5.3 Conclusions ....................................................... 136
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Matrix and Graph Datasets</td>
<td>55</td>
</tr>
<tr>
<td>3.2 Web Graph Datasets</td>
<td>56</td>
</tr>
<tr>
<td>3.3 Running time of PageRank (in seconds)</td>
<td>63</td>
</tr>
<tr>
<td>3.4 Running time of HITS (in seconds)</td>
<td>65</td>
</tr>
<tr>
<td>3.5 Average running time of Random Walk with Restart (in seconds)</td>
<td>67</td>
</tr>
<tr>
<td>5.1 Topics related to “world cup” from 5pm Jun 12th to 5pm Jun 13th.</td>
<td>113</td>
</tr>
<tr>
<td>5.2 Topics related to “world cup” from 5pm Jul 3rd to 5pm Jul 4th.</td>
<td>115</td>
</tr>
<tr>
<td>5.3 Topics related to “world cup” from 5pm Jul 4th to 5pm Jul 5th.</td>
<td>115</td>
</tr>
<tr>
<td>5.4 Event detection from the topics in Table 5.2 and 5.3</td>
<td>116</td>
</tr>
<tr>
<td>5.5 Top 20 ranked users about the keyword “travel”</td>
<td>123</td>
</tr>
<tr>
<td>5.6 Clustering results for the keyword “coupons”</td>
<td>129</td>
</tr>
<tr>
<td>5.7 Continue event for the keyword “justinbieber”</td>
<td>131</td>
</tr>
<tr>
<td>5.8 Merge event for the keyword “#madisonsquaregaga”</td>
<td>133</td>
</tr>
<tr>
<td>5.9 Split event for the keyword “#oilspliff”</td>
<td>137</td>
</tr>
<tr>
<td>6.1 Number of clusters.</td>
<td>148</td>
</tr>
</tbody>
</table>
6.2 Computation Times for the Back End. ............................. 149
6.3 Computation Time Comparison. ................................. 150
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Proposed Framework</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Temporal Snapshots at time t=1 to 6</td>
<td>22</td>
</tr>
<tr>
<td>2.2 Hardware Organization and Memory Hierarchy of a CUDA Device [82]</td>
<td>29</td>
</tr>
<tr>
<td>3.1 Illustrative example of partially tiling.</td>
<td>39</td>
</tr>
<tr>
<td>3.2 Illustrative example of composite storage.</td>
<td>45</td>
</tr>
<tr>
<td>3.3 SpMV kernels comparison on matrices representing power-law graphs.</td>
<td>58</td>
</tr>
<tr>
<td>3.4 SpMV kernels comparison on unstructured matrices [10].</td>
<td>60</td>
</tr>
<tr>
<td>3.5 Performance and bandwidth of PageRank.</td>
<td>62</td>
</tr>
<tr>
<td>3.6 Performance and bandwidth of HITS.</td>
<td>64</td>
</tr>
<tr>
<td>3.7 Performance and bandwidth of RWR.</td>
<td>66</td>
</tr>
<tr>
<td>3.8 Scalability of multi-GPU PageRank on web graphs</td>
<td>68</td>
</tr>
<tr>
<td>3.9 Auto-tuning: Auto vs Exhaustive searched number of tiles.</td>
<td>69</td>
</tr>
<tr>
<td>3.10 Auto-tuning: Auto vs Exhaustive searched performance.</td>
<td>70</td>
</tr>
<tr>
<td>3.11 Performance modeling: Predicted vs Measured performance.</td>
<td>70</td>
</tr>
<tr>
<td>4.1 Summarization via Pattern Utility and Ranking (SPUR) Framework</td>
<td>77</td>
</tr>
<tr>
<td>4.2</td>
<td>Division and compression of message stream</td>
</tr>
<tr>
<td>4.3</td>
<td>A batch of tweets compressed to a summary</td>
</tr>
<tr>
<td>4.4</td>
<td>Dynamic adjustment of pattern utility</td>
</tr>
<tr>
<td>4.5</td>
<td>Maintaining pyramidal time window</td>
</tr>
<tr>
<td>4.6</td>
<td>An example of the pyramidal time window</td>
</tr>
<tr>
<td>4.7</td>
<td>An example of the time series of a transaction</td>
</tr>
<tr>
<td>4.8</td>
<td>Decompose graph $G$ into $G_{infreq}$ and $G_{freq}$</td>
</tr>
<tr>
<td>4.9</td>
<td>Compress graph $G_{freq}$ into transaction set $T$ and pattern set $P$</td>
</tr>
<tr>
<td>4.10</td>
<td>Running time: SPUR vs. CDB vs. RPMine</td>
</tr>
<tr>
<td>4.11</td>
<td>False positive: SPUR vs. CDB vs. RPMine</td>
</tr>
<tr>
<td>4.12</td>
<td>Compression Ratio: SPUR vs. CDB vs. RPMine</td>
</tr>
<tr>
<td>4.13</td>
<td>Running time: SPUR vs. Krimp</td>
</tr>
<tr>
<td>4.14</td>
<td>Compression Ratio: SPUR vs. Krimp</td>
</tr>
<tr>
<td>4.15</td>
<td>Running time: stream summarization</td>
</tr>
<tr>
<td>4.16</td>
<td>Memory footprint: stream summarization</td>
</tr>
<tr>
<td>4.17</td>
<td>Query accuracy (T=62)</td>
</tr>
<tr>
<td>4.18</td>
<td>Query accuracy (T=30)</td>
</tr>
<tr>
<td>4.19</td>
<td>Compression ratio of G-SPUR on web and social network graphs.</td>
</tr>
<tr>
<td>4.20</td>
<td>CPU speed up of PageRank on compressed graphs.</td>
</tr>
<tr>
<td>4.21</td>
<td>GPU speed up of PageRank on compressed graphs.</td>
</tr>
<tr>
<td>5.1</td>
<td>Modification to the D-SPUR algorithm</td>
</tr>
</tbody>
</table>
5.2 G-SPUR compression of Network Topology ............... 119
5.3 Extract subgraph with G-SPUR .................................. 120
5.4 Topic volume of merge event. ................................. 135
6.1 Visualization of topic modeling result. ....................... 140
6.2 Visualization of event detection result. ....................... 141
6.3 Overview of proposed toolkit ................................. 143
6.4 Illustration of coarsened high-level view. .................... 151
6.5 Event View for Wikipedia ..................................... 154
6.6 Zoom feature .................................................. 157
6.7 Node View : Neighborhood of R. Agrawal .................... 159
6.8 Examples of a) Merge Event b) Split Event .................. 161
## List of Algorithms

1. Tile-Composite Kernel Auto-tuning .................. 50
2. Partition\( (T) \): Partition of one tile of matrix .............. 51
3. PM\( (T, WL) \): Performance Modeling of tile T given workload size .... 53
4. SPUR\( (B, M, \sigma, f) \) .................................. 80
5. Utility\( (p, f) \) ........................................... 84
6. UpdateRank\( (Q, p) \) ........................................ 86
7. Intersection\( (C_i, C_{i+1}) \) ................................. 148
Chapter 1: Introduction

Over the past decade, advances in technology have allowed us to collect and store huge amount of data from different domains, such as scientific research, engineering applications and social networks. Following the collection and storage of data, there is a great demand for analyzing and deriving valuable information out of the data. The field of Data Mining research tries to address the issues originated from the process of discovering knowledge from data. Data mining incorporates techniques from other research fields such as statistics, machine learning, high performance computing, database systems and so on. The challenge for Data Mining researchers is that of developing intelligent and efficient algorithms for deriving valuable knowledge from data.

Datasets generated from different real-world applications can be modeled as graphs, where nodes represent entities of interest and edges mimic the relationships among the entities. Examples of such datasets range from the World Wide Web to social networks, from protein-protein interaction networks to gene expression networks. The graph representations of such datasets are highly flexible and appropriate. The study of such graphs can help us understand the structural and behavioral patterns
originated from the datasets. There are several categories of algorithms targeting different aspects of knowledge or patterns from graph data. Next, we give an overview of important graph mining tasks and representative algorithms.

- **Link analysis:** How do we measure the authoritativeness of web pages in the World Wide Web? How do we measure the topological closeness or proximity between two nodes in a network? To answer such questions, the link analysis algorithms study graph datasets based on their link structure and produce a global ranking of the graph nodes to measure the authoritativeness, or compute a value for each pair of nodes to measure their proximity. The most important application of link analysis algorithms is the web search engine. The PageRank [21, 83] algorithm, introduced by Brin and Page, later became a commercial success as an integrated component of the Google search engine.

- **Graph clustering:** Graph clustering is also an important problem for data represented in the form of graphs. It tries to discover groups of nodes in the graph that are well connected within the group but are weakly connected to the nodes outside the group. There are many applications of graph clustering in various domains. For example, community detection in social networks, complex discovery in biological networks, clustering analysis in the World Wide Web and so on.

- **Dynamic Analysis:** Most datasets are changing in nature, either over time or in response to other factors. Dynamic analysis of graph data is an important graph mining topic in capturing the evolutionary behavior of graphs. Examples
of dynamic graph mining problems include modeling of evolutionary behaviors of dynamic graph, detecting critical events happened in dynamic graphs.

Social network analysis is an important real-world application of graph mining. As an example of social network websites, Twitter attracts a lot of the attentions from data mining researchers recently. We list some characteristics of the data originated from real-world applications such as Twitter. First of all, the network is usually heterogeneous by integrating information from multiple sources. For example, on Twitter, we have textual information from the messages written by its users, temporal information of the messages, topological information due to the follower-followee relationship among users and geographical information of the users. Second, the data is represented by high-speed streams instead of static or dynamic snapshots.

However, it is difficult to efficiently solve these graph mining problems in real-world applications and effectively present the results of these mining algorithms on large graph datasets for several reasons.

1.1 Challenges in Analyzing Large Graphs

**Scalability of large-scale graph mining algorithms:** The scale of modern graph datasets is usually in the order of millions of nodes and edges, for example, the Wikipedia article-article graph and the social networks such as Livejournal [75, 74]. The World Wide Web is even in the scale of billions of nodes and edges. Scalability is critical to the knowledge discovery process because it needs to deal with such large-scale input data. Furthermore, most graph mining algorithms are iterative in nature. This strengthens the need of fast solutions of these algorithms so that they can provide valuable information from these large datasets as *efficiently* as possible.
Storage of large-scale and streaming data: The graph datasets originated from real-world applications such as Twitter are streaming and heterogeneous in nature. The heterogeneous network data contains both the network content and topology generated by network users. Due to continuous user activities, the user generated data is in the form of a high speed data stream. The memory footprint of such data stream will grow linearly with time and can easily overflow the memory of a powerful computer system. It is challenging to efficiently store and manage such data with a budgeted memory space.

Complex query to discover new knowledge: It is desirable to support complex querying mechanism on the large-scale heterogeneous network data. Given a streaming and heterogeneous network, such as the Twitter social network, the design of complex queries should consider three key sources of information: network content, network topology and temporal information. It is challenging to design new queries that can take into account all three aspects of these information and discover new knowledge. It is also extremely challenging to efficiently process these queries.

Visualization of mining algorithms: Traditional data mining research focused on discovering interesting knowledge from data. Given the nature of the knowledge discovery process with a human-in-the-loop property, visual aids for presenting the mining results and the procedure of mining algorithms often play an important role in forming important insights from the data. There are several challenges in providing visualization support for graph mining problems. First, due to the large-scale of graph datasets and the limited space of computer screens, there must be coarsening and summarization mechanisms to present mining results. Second, multiple views should be provided to show different aspects of the graph mining procedure. Third,
the speed of the back-end mining algorithms should be fast enough to shorten the response time of the front-end visual interfaces.

The aforementioned challenges lead to the statement of this thesis.

**Thesis Statement:** We believe that the speed of analytical problems on large-scale network data can be accelerated by new GPU-based high performance computing technologies, and the storage size of the data can be reduced by data summarization techniques. Furthermore, novel query mechanisms coupled with effective visualization can enable the discovery of new knowledge.

### 1.2 Proposed Framework

![Proposed Framework](image)

Figure 1.1: Proposed Framework
To address these challenges, we propose a framework that facilitates the analysis of large-scale graph datasets. Figure 1.1 presents an overview of the proposed framework. We next describe the main components in detail.

- **High performance computing platform with GPUs**: The first component of our framework addresses the problem of scalability in graph mining applications. The objective here is to speed up the computational bottleneck of different graph mining algorithms. Of particular interest in this context, is to use modern GPU architectures to improve the efficiency of various mining algorithms. Here we choose to use the GPUs because they have a highly paralleled architecture with high memory bandwidth\(^1\), which is very suitable for data and computation intensive applications such as data mining. Then we implement a series of graph mining algorithms such as PageRank, HITS and Random Walk with Restart on top of the GPU computing platform. These algorithms share a common sparse matrix-vector multiplication (SpMV) kernel. The performance of this kernel on GPUs is much higher than on CPUs.

- **Summarization of large-scale streaming network content and topology data**: In real-world applications such as the Twitter social network, the data can be represented by high-speed streams with both network content and network topology information. Such input data can easily overflow the memory space of a computer system. To support such large-scale streaming and heterogeneous input data, we provide a summarization step to efficiently build an

---

\(^1\)The memory bandwidth on GPU is high, but the bandwidth of the memory bus between CPU and GPU is low.
in-memory summary of the input data in the second component of our framework. Further analytical tasks will be performed over the summary instead of the raw input data.

- **Complex query of network content and topology:** Our third component supports complex query of the summarized network content and topology to discover new knowledge from a large-scale streaming and heterogeneous network. The efficient processing of the queries depends on the high performance mining kernels and the compact storage of the data.

- **Visual-analytic front-end:** In our framework, we also provide a visual-analytic toolkit to facilitate the visual presentation of static and dynamic mining algorithms and results. This toolkit serves as an interface for the users to interact with mining algorithms. The interactive response time of the front-end is key to the usability of the toolkit. Hence, efficient back-end processing of the data is essential.

### 1.3 Our Contributions

We next highlight the key contributions of this thesis:

- **High performance mining kernels:** We identified a commonly used kernel, sparse matrix-vector multiplication (SpMV), from a series of mining algorithms including PageRank, HITS and Random Walk with Restart. We developed architecture conscious optimizations for speeding up this kernel on the modern
GPU architecture. We observed that the graph datasets of the mining algorithms present strong power-law characteristics, which lead to the low performance of computing the SpMV kernel on the GPU architecture. We developed corresponding optimizations by utilizing the texture cache on GPUs. Using real dataset, our optimizations can yield about 2x speedup over competing GPU methods and about 30x speedup over CPU methods. We also show that the optimized SpMV kernel translates to high performance in the graph mining algorithms that rely on this kernel. We also extend these optimizations to a cluster environment with multiple GPUs. This extension enables our high performance computing platform to handle out-of-core dataset which cannot fit in the memory of a single GPU. Moreover, the high performance of our optimized kernel relies on significant programmer expertise to tune parameters of the approach. To alleviate this, we developed a systematic mechanism for tuning the parameters automatically at runtime depending on the characteristics of the input data.

- **Memory efficient stream summarization of network content and topology:** To expand the application scope of our proposed framework, we proposed a memory efficient summarization method to build an in-memory summary of high speed streaming input. The input network data can contain both user generated content and topological information about user connections. The summary can be used as input to our large-scale analysis framework to perform analytical tasks. Experimental results show that our method can efficiently and incrementally summarize the stream data. The memory footprint of the summarization algorithm grows logarithmically with time instead of linearly.
The storage size of both the network content and topology can be significantly reduced. The raw network content data can be approximately reconstructed by querying the summary so as to support the analytical applications over the original data. The network topology can be losslessly reconstructed and graph mining algorithms on the summarized topology can run faster because of the compact storage.

- **Complex query of network content and topology:** We proposed new complex queries on the summarized network content and topology. The general format of the complex queries we proposed contains a query keyword of the network content and a query time interval. For analyzing network content, we reconstruct the messages that contain the query keyword in the time interval. We propose topic event detection queries to detect the evolution of the popular topics in two time intervals. For network topology, we project the entire network topology onto a subgraph using the query keyword and time interval. Then graph mining algorithms such as link analysis, graph clustering and event detection can be performed on the subgraphs. We show examples of query results from the Twitter social network data.

- **Interactive visual-analytic toolkit:** We developed a toolkit for visualizing static and dynamic graphs. Our toolkit provides multiple views to show multiple aspects of information from the data. The toolkit is designed with multi-level visualization of the graph data and supports the overview, zoom, filter and details-on-demand paradigm. To ensure scalability to large graphs, we have presented optimizations that can facilitate interactive visual analysis.
We have shown how the interactive features can aid the user in understanding information about static and dynamic networks in an effective and efficient manner.

1.4 Organization

In the next chapter we provide background and related work to each component of our framework. Chapter 3 – 6 address our work in each of the components of our proposed framework. Chapter 3 covers our work in speeding up data intensive mining kernels with modern GPUs. In Chapter 4 we explain our data summarization techniques for enabling our framework to handle large-scale streaming and heterogeneous data. In Chapter 5 we introduce new complex queries to support the discovery of new knowledge from the summarized data. Chapter 6 deals with our efforts in information visualization and interactive visual analysis. We conclude with Chapter 7 and discuss some future directions.
Chapter 2: Background and Related Work

This chapter contains information on the background knowledge and related work of our large-scale network analysis framework. First, we discuss three important categories of algorithms in the graph mining community: link analysis, graph clustering and dynamic graph analysis. Second, we introduce the background of the GPU architecture which is used in our high performance graph mining kernels. Third, we discuss the related work of speeding up the graph mining kernels with GPUs. Fourth, we talk about the related work of the data summarization component in our framework. Finally, we introduce the relevant work for visual analytics.

2.1 Graph Mining Algorithms

2.1.1 Link Analysis

Let $G = (V, E)$ denote the underlying graph of a dataset, $V$ is the node set and $E$ is the edge set. The link analysis algorithms take the adjacency matrix $A$ of the graph $G$ as input, where $A[u, v] = 1$ if there is a link from node $u$ to node $v$, and zero otherwise. The adjacency matrix contains all the link structure information from the graph. If we consider weighted graphs, other sources of information such as node attributes and similarities between two adjacent nodes can also be integrated into the
adjacency matrix. Next, we introduce two important mining tasks that link analysis algorithms can perform on such graph datasets.

**Link-Based Ranking:** Given the adjacency matrix of the graph, a popular analytical task we can perform is *link-based ranking* of the nodes in the graph, which is a primary focus of the link analysis community. The objective of link-based node ranking is to exploit the link structure of a graph to order or prioritize the nodes within a graph. The role of ranking is very important especially in the context of World Wide Web search. A query on the Web can have thousands of relevant results. If the ranking function of a search engine does not output what the user is looking for within the top range of the ranking, the effectiveness of the search engine is questionable. The task of ranking tries to give a ranked list of the nodes in the graph based on the *authoritativeness* of the graph nodes. The definition of an authoritative webpage can be described as a page with not only relevant information to the query, but it is also a *trusted* source of correct information. We consider the link information as a representation of the endorsement between two graph nodes. For example, if a web page $p$ points to a web page $q$, then we can assume that page $p$ endorses and recommends the content of page $q$. Therefore, we can think of the graph as a network of recommendations which contains information about the authoritativeness of the nodes. The task of ranking is to rank the nodes according to their authoritativeness. The ranking algorithm produces a weight for each node. The weight should be able to capture the authoritativeness of each node and it is used to rank the nodes. In the context of web information retrieval, the PageRank [21, 83] and HITS [61] algorithms are the most famous approaches to link-based node ranking.
Following the PageRank and HITS algorithms, a large number of modifications [90, 78, 66, 12, 37] as well as novel approaches [32, 19] were proposed. Bharat [12] and Chakrabarti [28] propose variants of HITS that exploit web page content to weight graph nodes and edges. Ng et al [78] discuss the stability of HITS and PageRank to perturbations of the link structure and present improvements to HITS that produce more stable rankings. Haveliwala [41] and Jeh et al [50] propose topic-sensitive ranking algorithms that emphasis the topically authoritative web pages. Ding et al [35] propose a unified framework by encompassing both PageRank and HITS as special cases and they propose several new ranking algorithms within this framework. Cohn [32] introduces a probabilistic model to HITS based on latent semantic indexing, where the model attempts to explain the link structure in terms of a set of latent factors. Richardson and Domingos [95] present probabilistic models that combine both content information and link structure.

**Graph Proximity:** Besides the task of producing a global ranking of all nodes in the graph, measurement of the proximity, relevance or closeness score between two nodes is another fundamental task in graph mining. In many real graphs, the connections between two nodes, e.g. the relationship between two persons in a social network, often presents complex behaviors. Traditional graph distance, e.g. shortest path, fails to capture such characteristics. To address this issue, Random Walk with Restart (RWR) [85, 106] is proposed as a basic measurement. The proximity score defined by Random Walk with Restart has many good properties: compared with pair-wise metrics, it can capture the global structure of the graph [42]; compared with traditional graph distances, such as shortest path, maximum flow etc, it can capture the multi-facet relationship between two nodes [106]. There are several measurements
of graph proximity similar to Random Walk with Restart, including electricity based method [123], graph based semi-supervised learning [36, 122] method and so on. Jeh and Widom [49] propose a metric for assessing the similarity of two graph nodes based on the degree to which they link to similar nodes. The similarity between two nodes in a directed or bipartite graph can be computed using a random walk formulation. Sun et al [105] propose a related object ranking approach for relevance search and anomaly detection that combines random walks and graph partitioning to improve scalability. Many applications take random walk and related methods as building blocks, such as personalized PageRank [41], content-based image retrieval [42], cross modal correlation discovery [85] and so on.

In the previous paragraphs, we surveyed two link mining tasks on graph data: link-based ranking and graph proximity. We introduced important algorithms of each task, PageRank and HITS for link-based ranking, and Random Walk with Restart for graph proximity. We also listed some related work of these algorithms. In the following sections, we will describe the details of the PageRank, HITS and Random Walk with Restart algorithms. We will show that all these three algorithms share a common SpMV kernel. This kernel is a very good candidate to be accelerated in our high performance mining framework.

PageRank

The PageRank algorithm models the link structure of web graphs by the random walk behavior of a random surfer [21, 83]. The web graph can be represented by a directed graph \( G = (V, E) \), where \( V \) is a set of \( n \) vertices and \( E \) is the set of directed edges. The adjacency matrix \( A \) is defined as \( A(u, v) = 1 \) if edge \( (u, v) \in E \); otherwise, \( A(u, v) = 0 \). Matrix \( W \) denotes the row normalized matrix of \( A \). The PageRank
vector $p$ is computed iteratively using the following equation until it converges:

$$p^{(k+1)} = cW^Tp^{(k)} + (1 - c)p^{(0)}$$  \hspace{1cm} (2.1)$$

where $c$ is a damping factor (set to 0.85 in our experiment), $p^{(0)}$ is initialized as a $n$ by 1 vector with all elements set to $1/n$. We can see that the major computational cost of Equation 2.1 is to compute the product of sparse matrix $W^T$ and vector $p^{(k)}$.

**HITS**

HITS is a link analysis algorithm of web pages [61]. It gives each web page two attributes: authority and hub. It rates web pages by assigning authority score and hub score to each web page. Let matrix $A$ be the adjacency matrix of a directed graph $G = (V, E)$ or $G$ may be a query specific subgraph of the whole web graph. Then the authority score vector $\vec{a}$ and hub score vector $\vec{h}$ are recursively defined as

$$\vec{a}^{(k+1)} = A^T\vec{h}^{(k)} \quad \vec{h}^{(k+1)} = A\vec{a}^{(k)}$$  \hspace{1cm} (2.2)$$

This recursive definition with two matrix-vector products can be rewritten as one matrix and vector multiplication by

$$\begin{bmatrix} \vec{a} \\ \vec{h} \end{bmatrix}^{(k+1)} = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \vec{a} \\ \vec{h} \end{bmatrix}^{(k)}$$  \hspace{1cm} (2.3)$$

Power method can be used to solve this eigen vector problem. Elements in $\vec{a}^{(0)}$ and $\vec{h}^{(0)}$ vectors are all initialized to $1/|V|$.

**Random Walk with Restart**

Random Walk with Restart (RWR) is an algorithm that tries to measure the relevance between two nodes in an undirected graph [85, 107]. Given a query node $i$
in the graph, the relevance score from all other nodes to node \( i \) forms a vector \( \vec{r}_i \). In RWR, vector \( \vec{r}_i \) is computed by the following equation:

\[
\vec{r}_i^{(k+1)} = cW\vec{r}_i^{(k)} + (1 - c)\vec{e}_i
\]

where \( c \) is a restart probability parameter (set to 0.9 in our experiment), \( W \) is the column normalized adjacency matrix and \( \vec{e}_i \) is a vector whose \( i^{th} \) element is 1 and all the other elements are 0. Vector \( \vec{r}_i \) can be computed using the power method.

From the formulas of these three link analysis algorithms, we can observe that they all rely on a common matrix and vector multiplication kernel to iteratively solve eigen vector problems with power methods. The matrices in these problems are the adjacency matrices of web graphs, so the matrices are very sparse and with power-law characteristics [31]. The high performance computing component of our framework, introduced in Chapter 3, will target at this SpMV kernel to accelerate link analysis algorithms such as the PageRank, HITS and Random Walk with Restart algorithms.

2.1.2 Graph Clustering

Most real-world interaction graphs exhibit modular structure [77]. The entire network can be partitioned into groups such that nodes within a group are well connected but are weakly connected to other groups. The problem of grouping well-connected nodes in a graph is known as graph clustering. The extraction of these groups or clusters can assist the understanding of properties of the underlying graph. Graph clustering has many applications in various domains, such as social networks, biological networks and the World Wide Web, to name a few. Many approaches have been proposed to find clusters in large graphs. In this section, we give an overview of some algorithms that are relevant to our work.
Metis

Metis is an algorithm for partitioning graphs into balanced clusters developed by Karypis et al [58]. It employs a hierarchical strategy to improve the efficiency of the algorithm. The Metis algorithm has three phases: coarsening, partitioning and refinement. In the coarsening phase, the original graph is transformed into a sequence of smaller graphs, forming a hierarchy. Next, an initial k-way partitioning of the coarsest graph is obtained. Note that this partitioning has to satisfy the balancing constraints as well as to minimize the object function of edge cuts. During the refinement phase, the partitioning on a higher level of the hierarchy is projected back to the lower level graphs. After projecting a partition, the refinement algorithm is used to reduce the edge cut while conserving the balance constraints. The Metis algorithm emphasizes the balance in cluster sizes, but in some domains such as bioinformatics and social networks, it is not necessarily true that the communities are with balanced size.

MCL

The MCL (Markov Clustering) algorithm proposed by Dongen [109] is an unsupervised graph clustering algorithm, based on the simulation of stochastic flows. The algorithm simulates the random walks within a network by alternating two operations called expansion and inflation. In the expansion step, the algorithm computes random walk probability at each graph node by expanding the stochastic flows to higher order neighbors (e.g. neighbors in two or higher hops). The probabilities associated with node pairs within the same cluster will be higher because there will be more paths to connect node pairs in a well-connected community. In the inflation step,
the algorithm will boost up the stochastic flows of intra-cluster walks and weaken inter-cluster walks. This is achieved by squaring and re-normalizing the transition probability of each node, without any a priori knowledge of the underlying cluster structure. Eventually, the iterations of the expansion and inflation operations result in the separation of the graph into different partitions.

The MCL algorithm is widely used in bioinformatics applications. Several graph clustering algorithms are empirically compared on protein-protein interaction networks to identify protein complexes [23]. The clusters obtained are compared with known annotated complexes. The conclusion of this study is that the MCL algorithm outperforms the other algorithms in the task of extracting complexes from interaction networks.

**MLR-MCL**

However, the MCL algorithm has its limitations. First of all, it tends to fragment communities by segmenting well connected clusters into small clusters. Second, the MCL algorithm cannot scale to large graphs. The basic computations in the algorithm are performed by iteratively take the product of two large sparse matrices. Satuluri [99] proposed the MLR-MCL (Multi-Level Regularized MCL) algorithm to address the weakness of MCL while retaining it strengths. Regularized-MCL (R-MCL) is proposed to alleviate the fragmentation problem. A scalable multi-level strategy, similar to Metis, is used to speed up the R-MCL algorithm, called MLR-MCL. Empirical studies show that MLR-MCL algorithm improves the MCL algorithm in both clustering quality and speed. The major computational cost of the MLR-MCL
algorithm is a sparse matrix-matrix multiplication kernel. It is very difficult to efficiently speed up this kernel with parallel computing platforms, because the kernel performance is highly dependent on the non-zero distributions in the matrix.

**Local Spectral Clustering**

Local spectral clustering [4] algorithm attempts to find a cluster of nodes in a local region near a given seed node, without examining the entire graph. The running time of local clustering algorithm is proportional to the size of the discovered cluster, instead of the size of the entire graph. So the algorithm can scale to graphs with billions of nodes and edges. Additionally, this algorithm can discover multiple and potentially overlapping clusters by initiating the algorithm from multiple seed nodes. The local spectral clustering algorithm starts with a random walk from the seed node, which is equivalent to the Personalized PageRank [41] or the Random Walk with Restart [85, 106] algorithms. Intuitively, if a node belongs to a well-connected community, the random walk will end up with higher probabilities at other nodes that are in the same community. Therefore, nodes in the same community as the seed node will have higher Personalized PageRank values than nodes in the other communities. The local spectral clustering algorithm computes Personalized PageRank iteratively from the seed node. With more iterations, the PageRank values will spread to more nodes in the graph and make the algorithm non-local. The local spectral clustering algorithm truncates the PageRank vector at the end of each iteration by only keeping the nodes with high PageRank values. Until the algorithm converges, the local community around the seed node is found.

The major computational cost of the local spectral clustering algorithm is the computation of the Personalized PageRank vector from the seed node. It also depends
on an iterative SpMV kernel the same as the Random Walk with Restart algorithm introduced in Section 2.1.1. Therefore, it can be accelerated by our GPU based SpMV kernel in our high performance analytics framework introduced in Chapter 3.

2.1.3 Dynamic Graph Analysis

The studies introduced in previous sections have focused on mining static graphs to identify patterns, community structures and novel information. However, the dynamic behaviors of clusters and communities in graphs have attracted the interest of graph mining researchers recently.

Leskovec et al [68] studied the evolution of graphs based on topological observations, such as the degree distribution and graph diameters of large networks. They empirically showed that these networks become denser over time, with the densification following a power-law pattern. Backstrom et al [6] studied formation of groups and the patterns of their growth and evolution over time. Chi et al [29] proposed evolutionary settings for the spectral clustering algorithm by incorporating temporal smoothness to obtain consistent clusters.

These works assume that the dynamic graph data is represented by a series of graph snapshots. Alternatively, a dynamic graph can also be in the form of an edge stream, which indicates the updates to the dynamic graph. Under this streaming model of dynamic graphs, Aggarwal et al [2] propose a probabilistic approach to build a summary of the streaming edges and they demonstrate that the summary can be used to mine frequent dense patterns in the dynamic graphs. Becchetti et al [9] develop a streaming algorithm to approximately count the triangles in the dynamic
graphs. Henderson et al [44] propose a framework to detect abnormal behaviors from high-speed edge streams by monitoring several key metrics of the dynamic graphs.

With a brief description of related research in dynamic graph analysis, we next introduce an event detection algorithm on snapshots of dynamic graphs proposed by Asur et al [5], which is studied in our proposed framework.

**Event Detection**

The event detection framework proposed by Asur et al [5] is focused on identifying important events and behavioral patterns that can characterize and model the evolutionary trends of dynamic graphs. We next introduce the formal definition of critical events. The events defined are primarily between two consecutive snapshots of a dynamic graph. The events are divided into two categories: events involving communities and events involving individuals.

Figure 2.1 displays a set of snapshots of the network which will be used as a running example in this section. At time $t = 1$, 2 clusters are discovered (shown in different colors).

**Events involving communities:** 5 basic events which clusters can undergo are defined between any two consecutive time intervals or steps. Let $S_i$ and $S_{i+1}$ be snapshots of $S$ at two consecutive time intervals with $C_i$ and $C_{i+1}$ denoting the set of clusters respectively. The five proposed events are:

1) **Continue:** A cluster $C_{i+1}^j$ is marked as continuation of $C_i^k$ if $V_{i+1}^j$ is the same as $V_i^k$. 
Figure 2.1: Temporal Snapshots at time t=1 to 6

\[
Continue(C_i^k, C_{i+1}^j) = 1 \text{ iff } V_i^k = V_{i+1}^j
\]

The main motivation behind this is that if certain nodes are always part of the same cluster, any information supplied to one node will eventually reach the others. Therefore, as long as the vertex set remains same, the information flow is not hindered. The addition and deletion of edges merely indicates the strength between the nodes. An example of a continue event is shown at t=2 in Figure 2.1. Note that an extra interaction appears between the nodes in Cluster \( C_2^1 \) but the clusters do not change.

2) \( \kappa \)-Merge: Two different clusters \( C_i^k \) and \( C_i^l \) are marked as merged if there exists a cluster in the next timestamp that contains at least \( \kappa \% \) of the nodes belonging to these two clusters. The essential condition for a merge is:
\[ \text{Merge}(C^k_i, C^l_i, \kappa) = 1 \text{ iff } \exists C^j_{i+1} \text{ such that} \]
\[
\frac{|(V^k_i \cup V^l_i) \cap V^j_{i+1}|}{\text{Max}(|V^k_i \cup V^l_i|, |V^j_{i+1}|)} > \kappa\% \quad (2.5)
\]

and \[ |V^k_i \cap V^j_{i+1}| > \frac{|C^k_i|}{2} \] and \[ |V^l_i \cap V^j_{i+1}| > \frac{|C^l_i|}{2} \]. This condition will only hold if there exist edges between \( V^k_i \) and \( V^l_i \) in timestamp \( i + 1 \).

Intuitively, it implies that new interactions have been created between nodes which previously were part of different clusters. This caused \( \kappa\% \) of nodes in the two original clusters to join the new cluster. Note that, in an ideal or complete merge, with \( \kappa = 100 \), all nodes in the two original clusters are found in the same cluster in the next timestamp. The two original clusters are completely lost in this scenario.

We use the second term in the denominator in the above equation to differentiate cases where the merged cluster is very large and in which case the identities of the original clusters are lost. These do not qualify as \textit{Merge} events.

Figure 2.1 shows an example of a complete merge event at \( t=3 \). The dotted lines represent the newly created edges. All the nodes now belong to a single cluster \((C^1_3)\).

3) \( \kappa\)-\textit{Split}: A single cluster \( C^j_i \) is marked as split if \( \kappa\% \) of nodes from this cluster are present in 2 different clusters in the next timestamp. The essential condition is that:

\[ \text{Split}(C^j_i, \kappa) = 1 \text{ iff } \exists C^k_{i+1}, C^l_{i+1} \text{ such that} \]
\[
\frac{|(V^k_{i+1} \cup V^l_{i+1}) \cap V^j_i|}{\text{Max}(|V^k_{i+1} \cup V^l_{i+1}|, |V^j_i|)} > \kappa\% \quad (2.6)
\]

and \[ |V^k_{i+1} \cap V^j_i| > \frac{|C^k_{i+1}|}{2} \], \[ |V^l_{i+1} \cap V^j_i| > \frac{|C^l_{i+1}|}{2} \].

\(^2\)We used \( \kappa \) values of 30 and 50 in our experiments.
Intuitively, a split signifies that the interactions between certain nodes are broken and not carried over to the current timestamp, causing the nodes to part ways and join different clusters. Also note that a broken edge, by itself, does not necessarily indicate a split event, as there may be other interactions existing between vertices in the cluster (similar to the notion of k-connectivity). Time t=4 in Figure 2.1 shows a split event when a cluster gets completely split into three smaller clusters.

4) **Form:** A new cluster $C_{k+1}^i$ is said to have been formed if none of the nodes in the cluster were grouped together at the previous time interval i.e. no 2 nodes in $V_{k+1}^i$ existed in the same cluster at time period $i$.

\[ \text{Form}(C_{i+1}^k) = 1 \text{ iff } \exists \text{ no } C_i^j \text{ such that } V_{i+1}^k \cap V_i^j > 1 \]

Intuitively, a form indicates the creation of a new community or new collaboration. Figure 2.1 at time t=5 shows a form event when two new nodes appear and a new cluster is formed.

5) **Dissolve:** A single cluster $C_i^k$ is said to have dissolved if none of the vertices in the cluster are in the same cluster in the next timestamp i.e. no two entities in the original cluster have an interaction between them in the current time interval.

\[ \text{Dissolve}(C_i^k) = 1 \text{ iff } \exists \text{ no } C_{i+1}^j \text{ such that } V_i^k \cap V_{i+1}^j > 1 \]

Intuitively, a dissolve indicates the lack of contact or interactions between a group of nodes in a particular time period. This might signify the breakup of a community or a workgroup. Figure 2.1 at time t=6 shows a dissolve event when there are no longer interactions between the three nodes in Cluster $C_3^1$ resulting in a breakup of
the cluster into 3 clusters - $C_6^1$, $C_6^2$ and $C_6^3$.

Events involving individuals: We wish to analyze not only the evolution of communities but the influence of the behavior of individuals on communities. In this regard, we introduce four basic transformations involving individuals over snapshots.

1) **Appear**: A node is said to appear when it occurs in $C_i^j$ but was not present in any cluster in the earlier timestamp.

$$\text{Appear}(v, i) = 1 \text{ iff } v \notin V_{i-1} \text{ and } v \in V_i$$

This simple event indicates the introduction of a person (new or returning) to a network. In Figure 2.1, at time $t = 5$ two new nodes appear in the network.

2) **Disappear**: A node is said to disappear when it was found in a cluster $C_{i-1}^j$ but is not present in any cluster in the timestamp $i$.

$$\text{Disappear}(v, i) = 1 \text{ iff } v \in V_{i-1} \text{ and } v \notin V_i$$

This indicates the departure of a node from a network. In Figure 2.1, at time $t = 6$ two nodes of cluster $C_{5}^4$ disappear from the network.

3) **Join**: A node is said to join cluster $C_i^j$ if it exists in the cluster at timestamp $i$. This may be due to an *Appear* event or due to a leave event from a different cluster. Note that in case, the cluster $C_i^j$ must be sufficiently similar to a cluster $C_{i-1}^k$.

$$\text{Join}(v, C_i^j) = 1 \text{ iff } \exists C_i^j \text{ and } C_{i-1}^k \text{ such that } C_{i-1}^k \cap C_i^j > \frac{|C_{i-1}^k|}{2} \text{ and } v \notin V_{i-1} \text{ and } v \in V_i^j$$
The cluster similarity condition ensures that $C^j_i$ is not a newly formed cluster. This condition differentiates a Join event from a Form event. Nodes forming a new cluster will not be considered to be Join events since there will be no cluster $C^k_{i-1}$ in the previous timestamp with similarity $> \frac{|C^k_{i-1}|}{2}$ with the newly formed cluster.

4) **Leave**: A node is said to leave cluster $C^k_{i-1}$ if it no longer is present in a cluster with most of the nodes in $V^k_{i-1}$. A node that leaves a cluster may leave the network as a Disappear event or may join a different cluster. In a collaboration network, a Leave event might correspond to a student graduating and leaving a group.

$$Leave(v, C^j_i) = 1 \text{ iff } \exists C^j_i \text{ and } C^k_{i-1} \text{ such that } C^k_{i-1} \cap C^j_i > \frac{|C^k_{i-1}|}{2} \text{ and } v \in V^k_{i-1} \text{ and } v \notin V^j_i$$

The similarity constraint between the two clusters is used to maintain cluster correspondence. Note that if the original cluster dissolves, the nodes in the cluster are not said to participate in a Leave event. This is due to the fact that there will no longer be a cluster with similarity $> \frac{|C^k_{i-1}|}{2}$ with the dissolved cluster $C^k_{i-1}$.

### 2.2 GPU Background

GPUs have been used for accelerating graphics rendering pipelines before NVIDIA introduced the CUDA general purpose parallel computing architecture [70, 79]. CUDA provides an interface for the developers to program the devices for general purpose computations and data intensive applications. In this section, we discuss the hardware architecture and the programming model of CUDA GPUs. Figure 2.2 illustrates the organization of the computing hardware and the memory hierarchy in CUDA GPUs.
A CUDA device consists of a set of streaming multiprocessors (SMs), each one equipped with one instruction unit and a cluster of 8 streaming processors (SPs). The parallel region of a CUDA program is partitioned into a grid of thread blocks that run logically in parallel. The programmer can decide the dimensions of the grid and the block. Thread blocks are distributed evenly on the multiprocessors. A warp is a group of 32 threads that run concurrently on a multiprocessor. The execution of the threads follows a single instruction multiple threads (SIMT) model [70, 79]. The instruction unit on a multiprocessor issues one instruction for all the threads in the same warp at each time [82]. The streaming processors execute this instruction for all the threads in the warp. Different warps within a block are time-shared on the hardware resources. A kernel is the code in the parallel region to be executed by each thread. Conditional instructions cause a divergence in the execution if threads in the same warp take different conditional paths. The threads are serialized in this situation.

There are various memory units on a CUDA device. The device memory, which is also called the global memory, is a large memory which is visible to all threads on the device [82]. The access latency of the global memory is high. Memory requests of a half warp (16 threads) are served together at a time. When accessing a 4- or 8-byte word, the global memory is organized into 128-byte segments [82]. The number of memory transactions executed for a half warp is the number of memory segments requested by this half warp. The requests from the threads in the half warp are coalesced into one memory transaction if they are accessing addresses in the same segment. When the addresses accessed by the half warp are all in one segment, this

3Devices with Compute Capability lower than 1.2 have stricter requirements.
request is fully coalesced. The global memory is divided into 8 equally-sized memory partitions of 256-byte width. Concurrent memory requests to the global memory by all the active warps should be distributed uniformly amongst partitions. The term partition camping [81] is used to describe the situation when global memory accesses are congested and queued up at some partitions while the other partitions are idle. While coalescing concerns global memory accesses within a half warp, partition camping concerns global memory accesses amongst all active half warps. Each multiprocessor is equipped with an on-chip scratchpad memory [82], which is called the shared memory. The shared memory has very low access latency. It is only visible to the threads within one block and has the same lifetime as the block [82]. The shared memory is organized into banks. If multiple addresses in the same bank are accessed at the same time, it leads to bank conflicts and the accesses are serialized. There are also a set of registers shared by the threads in the block. The constant and texture memories are read-only regions in the global memory space with on-chip caches. The programmer can bind a region of the global memory to either the constant or the texture memory before the kernel starts.

2.3 Sparse Matrix and Vector Multiplication

Many graph mining algorithms share a common computational bottleneck – the sparse matrix-vector multiplication (SpMV) kernel. The SpMV kernel computes a vector $y$ as the product of a $n$ by $m$ sparse matrix $A$ and a dense vector $x$. Since the SpMV kernel is widely used in scientific computing, several existing efforts have targeted optimizing this kernel for the GPU [10, 8, 30]. However, none of the above
take into account the skew of the non-zero distribution present in matrices representing power-law graphs. The performance of the previous work is low on such matrices due to their power-law characteristics.

Bell and Garland [10] propose several representations of sparse matrices on the CUDA platform for SpMV kernels in NVIDIA’s SpMV library. The compressed sparse row (CSR) format is widely used to represent sparse matrices. In the CSR format, the non-zeros in the same row are stored contiguously in memory, and all rows are stored in one data array, with another array holding the column indices of the non-zeros. A third array of row pointers marks the boundary of each row. The corresponding CSR kernel from Bell and Garland [10] assigns the computation of each row to a thread. With power-law graphs, it is possible to balance the workload among thread blocks, but it is hard to balance among threads within one block. So all the threads
in one block will wait for the thread which is assigned to the longest row in this block. To optimize this method, Bell and Garland [10] developed the CSR-vector format, in which a warp of 32 threads are assigned to work on each row. This strategy only helps the rows with more than 32 non-zeros, but most of the nodes in power-law graphs have degree lower than 32. The computation resources of the warps assigned to such rows will therefore be wasted. Baskaran and Bordawekar [8] further optimized the CSR-vector format by using a half warp for each row to improve global memory accesses, and also a padding technique is used to ensure the memory requests are fully coalesced.

Besides the CSR format, the coordinate (COO) and ell-pack (ELL) formats are also used in Bell’s SpMV kernel. In COO format, all the non-zeros in matrix $A$ are combined in a long vector grouped by row index, and the kernel first computes the multiplication of each non-zeros with the corresponding elements of vector $x$ in the first pass; then the segmented reduction of the rows is done on this long vector by thread warps. In the reduction phase, because the length of each row is not necessary a multiple of warp size, synchronization points are heavily used and warp thread divergence is frequent. However, the COO kernel is the most insensitive to variable row length in the matrix according to the previous study [10]. The ELL format requires the number of non-zeros on each row is bounded by some small number $k$, so that the matrix $A$ can be represented by a dense $n$ by $k$ matrix $M$, in which only non-zeros in $A$ are stored, and the corresponding column indices of these non-zeros are also stored in a separate matrix. In the ELL kernel, $M$ is stored in column major, and the thread assigned to each row can access global memory very efficiently. In $M$, zeros are added to rows with fewer than $k$ non-zeros, so $k$ cannot be large,
otherwise it will introduce large overhead to access these zeros. The ELL format cannot be directly applied to graph mining algorithms, where the node degree in the graph cannot be bounded by a small number $k$. However, ELL and COO format can be mixed together to represent a matrix, where the first $k$ non-zeros of each row are stored in ELL format and the others are stored in COO format. This is the hybrid (HYB) kernel of NVIDIA’s SpMV Library [10]. There are two other formats in NVIDIA’s SpMV Library [10]. The diagonal (DIA) format is only applicable to matrices in which all non-zeros fall into a band around the diagonal. The packet (PKT) format first uses Metis [59] to cluster non-zeros into dense sub-blocks, then a sub-block is loaded into shared memory and processed by a thread block as a dense sub-matrix. Choi et al [30] propose blocked ell-pack format in which the non-zeros are stored in fixed size blocks first and the blocks are indexed with similar method in ELL format. Blocking techniques [98, 97] will gain locality when accessing vector $x$ and reduce loop overhead when computing matrix indices; but it will also introduce memory overhead if the small blocks cannot be filled with enough non-zero elements.

Because of the importance of the SpMV kernel, researchers have put substantial efforts on optimization techniques over various architectures and platforms. Vuduc et al [111, 110, 64] study optimizations and performance auto-tuning in single core CPUs over the Sparsity framework [48]; Nishtala [80] provides detailed research about how blocking can benefit SpMV kernel over CPU. Williams [115] compares SpMV kernels on emerging multicore platforms, including multicore CPUs and the Cell Broadband platform. Blelloch [14, 15] studies sparse matrix computation on vector machines. Sengupta [101] develops efficient segmented scan primitives for GPUs, which can be used in the reduction step of COO kernel. NVIDIA’s SpMV library implements a
more efficient segmented reduction than segmented scan. Recent work by Kang et al. [56] employs the MapReduce [34] framework to implement iterative SpMV kernel based on Hadoop, an open source version of MapReduce, and performs large-scale graph mining over this platform.

Previous studies emphasize performance optimization of the SpMV kernel on various platforms. However, they require parameter tuning to achieve high performance, which depends on the characteristics of the input matrices. Therefore, an important issue is the development of a performance model to guide the tuning. Hong [46] proposed an analytical model of GPU architecture based on the parallelism in memory accesses and computation. Ryoo [96] studied parameter space pruning on GPUs. An adaptive performance modeling tool based on the work flow graph of a GPU application is proposed by Baghsorkhi [7]. Choi [30] proposed a model-driven autotuning framework in their SpMV work.

A large class of graph mining algorithms leverage the SpMV kernel iteratively to perform computation until the algorithms converge, e.g. PageRank [21, 83], HITS [61] and Random Walk with Restart [85, 106, 107]. These algorithms first transform the adjacency matrix of a graph and then operate on the transformed matrix. The graph dataset used by these algorithms usually have strong power-law properties, hence the number of non-zeros on each row or column of the corresponding matrix will follow a power-law distribution. The skewness of the distribution leads to poor load balancing and low memory access efficiency on GPU.
2.4 Summarization of network content and topology

Since the network content generated by social media is available as a stream of time-stamped messages, dynamic studies have been conducted to track topics on the stream [69, 67], and find temporal patterns [118]. However, all of these studies assume unlimited amount of memory to store the data. We are interested in summarizing the data stream with a limited memory budget and serving new analytical tasks with the summary. Existing work can also benefit from the summary from approximately reconstructing the original data by querying the summary.

We treat each social media post as a transaction of words and use frequent patterns to summarize. The CDB [116] algorithm uses rectangles to cover a transactional database; RPMine [117] first tries to cluster the patterns and use the cluster centers to cover the remaining patterns. Methods that use a probabilistic model of the patterns [112] are slow and not capable of processing streams that arrive at a fast pace. All the above algorithms do not support mining on data streams. Krimp [103] follows the MDL principle to summarize the data with codetables, which is essentially a static ranking of patterns. StreamKrimp [65], the streaming version of Krimp, can dynamically adjust the codetable for streaming data. Traditional methods of mining frequent patterns on streams [26, 73] focus on either counting item or pattern frequencies rather than summarizing the data.

To summarize network topology, Boldi and Vigna [16, 18] proposed an encoding method to compress web graphs. Their encoding method can achieve high compression ratio on large-scale power-law webgraphs. However, the compressed data cannot be directly used by graph mining algorithms such as PageRank. The decompression process will take long time and large storage space. Buehrer and Chellapilla [25] used
virtual nodes to replace large and dense communities to compress the edges of in large web graphs. Karande and Chellapilla [57] extend the application of the virtual node method to speed up the PageRank and SALSA algorithm using the compressed data. But their methods did not leverage the high performance of modern parallel architectures like GPUs.

2.5 Visual Analytics

New visualization techniques and mining algorithms have been proposed to show patterns in network and graph data. Wang et al [114] proposed the CSV plot to detect and visualize cohesive communities of graph nodes. Zhang and Parthasarathy [121] use triangle cores to find and visually present dense areas of graphs. Graph drawing is a general technique to visualize topological structure of graphs. Toolkits including Prefuse 4 and GraphViz 5 use several energy-based graph drawing algorithms. However, these algorithms cannot scale well on large graphs. Algorithms to visualize internet-scale graphs were proposed by North et al [38]. But such visualizations are slow to compute and also need special monitors to display.

Henderson et al [45] developed MetricForensics to find and visualize patterns from large streaming graphs. Osprey [20] and Visant [47] are tools to visualize and analyze biological networks. There has been considerable amount of work in visualization of social networks. Heer and Boyd [43] have developed the Vizster tool for visualizing online social networks. The authors use a graph representation for visualizing data collected from the Friendster online community. The toolkit can be used to explore communities, linkage and supports keyword search. Perer and Schneiderman [88]

4http://prefuse.org
5http://www.graphviz.org
have presented a general social network visualization toolkit. The toolkit supports ranking of nodes based on various properties of the graph like centrality, cut-points etc. Abello and others [1] have presented a graph visualization toolkit called ASK–GraphView. The toolkit uses a clustering algorithm to construct a hierarchy which is easy to browse. Henry and Fekete [?] have presented a dual representation for visualizing social networks. The proposed toolkit MatrixExplorer uses a synchronized graph and matrix representation of the network for visualization. A key difference separating our work from the above methods is that they are designed to operate primarily on static interaction graphs.

Gloor and Zhao [40, 39] have developed iQuest, a visual toolkit to understand topics of discussion among actors in a semantic web. Kumar and Garland [63] have presented algorithms to visualize a graph in hierarchical fashion by exploiting existing correlations. Time-varying graphs are handled by producing animations composed of static snapshots. Qeli et al. [89] have proposed algorithms to visualize time-varying matrices. The matrices used in the article represent clustering results. The authors generate a cumulative matrix and use colors to denote changes in memberships. The toolkit can also be used to find a group of elements which are part of the same cluster for an extended period of time.
Chapter 3: High Performance Mining Kernels

In this chapter, we describe how to speed up data intensive mining kernels with modern GPUs. Over the last decade we have witnessed a revolutionary change in the way commodity processor architectures are being designed and implemented. CPUs with superscalar out-of-order execution, vector processing capabilities, and simultaneous multithreading, chip multiprocessing (CMP), and high-end graphics processor units (GPU) have all entered the mainstream commodity market. The new supercomputer system announced in China recently has set a performance record of 2.5 petaflops measured by the LINPACK benchmark, making it the fastest system in the world today. This system exemplifies modern heterogeneous computing by coupling massively parallel GPUs with multi-core CPUs. Data intensive algorithms often require significant computational resources, and thus stand to benefit significantly from such innovations if appropriately leveraged.

Previous studies [53, 54] proposed a generalized middleware for parallel data mining applications and addressed the issues of speeding up data mining algorithms on multi-core CPUs [92, 52], GPUs [71] and heterogeneous platform with CPUs and GPUs [93]. However, they are mostly targeting traditional data mining algorithms on transaction dataset, for example, k-means, EM and PCA algorithms. In our study, we are interested in accelerating graph-based mining algorithms with GPUs. In this
chapter we develop a novel approach [120] to facilitate the efficient processing of key
graph-based data mining algorithms such as PageRank [21, 83], HITS [61] and Ran-
dom Walk with Restart [85, 107] on modern GPUs. A common feature of these algo-
rithms is that they rely on a core sparse matrix vector multiplication kernel (SpMV).
Implementations of this kernel on GPUs have received much attention recently from
the broader scientific and high performance computing communities [24, 8, 30] in-
cluding an industrial strength effort from NVIDIA research [10]. We present a novel
non-parametric, self-tunable, approach to data representation for computing this ker-
nel, particularly targeting sparse matrices representing power-law graphs. Using real
web graph data, we show how our representation scheme, coupled with a novel tiling
algorithm, can yield significant benefits over the current state of the art GPU efforts
on a number of core data mining algorithms such as PageRank, HITS and Random
Walk with Restart.

The key difference between past work and ours is that here we are interested in
the processing of sparse matrices that represent large graphs – typically with power-
law [31] characteristics. This difference is also central to the specific architecture-
conscious approach we propose for processing the SpMV kernel. We transform and
represent the matrix in such a way so as to facilitate tiling – a key strategy used to
enhance temporal locality. Additionally we rely on a composite storage algorithm that
leverages the skew in the degree distribution afforded by the fact that these matrices
represent power-law graphs. Architectural features of the GPU such as the texture
cache are also effectively leveraged in the processing of the kernel. Moreover, we also
demonstrate how the basic approach can be extended to handle web-scale graphs with
billions of edges that do not fit in the memory of a single GPU by suitably leveraging multiple GPUs.

The method summarized above relies on significant programmer expertise to tune two key parameters of the approach. To alleviate this we present a systematic mechanism for tuning the parameters automatically at runtime depending on input matrix characteristics. Unlike existing work on parameter auto-tuning for SpMV kernel [30, 111], an important side effect of our approach is a reliable performance model for predicting overall performance of the kernel.

We present a comprehensive empirical evaluation of the proposed methods on three data mining algorithms and the base SpMV kernel on a range of real datasets including several web-scale graph datasets. We find that on moderately sized datasets that fit on a single GPU, our SpMV kernel as well as the graph mining methods that rely on this kernel – HITS, PageRank and Random Walk with Restart – are typically **1.8 to 2.1** times faster than an industrial strength state-of-the-art GPU competitor and anywhere from **18 to 32** times faster than a similarly structured and optimized CPU implementation. We find that our methodology can *scale quite comfortably to webscale datasets with billions of edges*, on multiple GPUs with parallel efficiencies of up to **70%**. Finally, we empirically demonstrate the effectiveness of our autotuning approach both in terms of its *ability to correctly select parameters for our kernel* on a wide range of datasets, and in terms of its ability to *reliably predict the absolute performance of the kernel* under different parametric settings, suggesting it can be used for adaptive algorithm designs in next generation hybrid architectures[87].
3.1 Methodology

In this section we first present optimization techniques for SpMV kernel on power-law matrices within a single GPU. Next, we show how our SpMV kernel can be extended to handle out-of-core matrices within a multi-GPU cluster. Finally, we present a systematic mechanism for automatically tuning the parameters in our optimization methods. We assume that the readers are familiar with the CUDA architecture. Details about CUDA are provided in Section 2.2.

![Illustrative example of partially tiling](image)

Figure 3.1: Illustrative example of partially tiling.

3.1.1 Single-GPU SpMV

Our optimizations are based on a series of observations from benchmarks that demonstrate the limitations of previous work. We propose solutions that target these limitations and thereby improve the performance.

The SpMV kernel is a bandwidth limited problem since the floating point operations per memory access is low. When computing the product of a sparse matrix $A$ and a vector $x$, the memory accesses to matrix $A$ are optimized to be fully coalesced
in NVIDIA’s SpMV library. But the accesses to vector \( x \) have never been optimized. Also vector \( x \) is the only reusable memory in the SpMV kernel.

**Observation 1: Each row accesses random elements in vector \( x \).** In the adjacency matrix of a power-law graph, the column indices of the non-zeros on each row are not continuous, and are relatively random, which leads to non-coalesced memory addresses when accessing \( x \). Previous work [10, 8] has bound the entire vector \( x \) to the texture memory and utilizes the cache of texture memory to improve the locality. But the size of \( x \) is usually much larger than the size of the texture cache. The resultant cache misses reduce memory bandwidth utilization due to the long latency of non-coalesced global memory accesses.

**Solution 1: Tiling matrix \( A \) and vector \( x \) with texture cache.** Suppose we divide matrix \( A \) into fixed width tiles by column index and segment vector \( x \) correspondingly, so that each tile of \( A \) only needs to access one segment of \( x \). If one segment of \( x \) can fit in the texture cache, once the elements are fetched into the cache, none of them will be kicked out until the computation of this tile finishes. Therefore, we can get maximum reuse of \( x \).

The texture cache size is a key factor in determining the width of the tiles. To estimate the texture cache size (since this is not provided by the manufacturer) on our Tesla GPU, we conduct benchmarking experiments as follows. We use a large sparse matrix and multiply it with a vector. The column indices are \( mod \)-ed by tile width, so all accesses to vector \( x \) are mapped to one tile. We vary the tile width from 100\( K \) to 1\( K \) and run the multiplication. The performance improves most significantly.
when tile width = 64K, corresponding to 256 KB of cache size. So our tile width is fixed to 64K columns.

The performance of tiling the entire matrix $A$ and vector $x$ is still low. Since we divide all the columns of matrix $A$ into tiles, there could be too many tiles when the matrix is large. Each tile needs to add its partial result to the final result $y$. Rows in each tile may become empty after tiling operation, leading to non-coalesced memory accesses to $y$ because we only need to write the result of non-empty rows back. Also, the write-back result of one tile has to be visible to the next tile before the next tile can start; otherwise, memory read-after-write conflicts could happen. To avoid such memory conflicts, we restart a CUDA kernel for each tile, which also causes an overhead. Therefore, tiling the columns of the matrix $A$ fails to improve the performance of the SpMV kernel.

**Observation 2: Column lengths follow a power-law distribution.** Suppose a matrix is the adjacency matrix of a power-law graph, the number of non-zeros in the columns of the matrix will follow a power-law distribution. So there are a large number of columns with few non-zeros. In tiles containing such columns, we cannot get much reuse of vector $x$, but we still need to restart a large number of kernels to compute them and incur the overheads. On the other hand, the long columns in the power-law distribution concentrates large portion of the non-zeros in the matrix. There is a lot of memory reuse of vector $x$ in such columns. The denser the columns, the more the benefits. If we can first tile such columns greedily, we can finish the
majority of the total computation efficiently in the first few tiles. The overall performance of the entire matrix will be improved.

**Solution 2: Reorder columns by column lengths and partially tile A.** Our idea is to first reorder the matrix columns in decreasing order of length. We can divide the reordered matrix into two sub-matrices by setting a threshold on column length. Long columns form a denser sub-matrix; the remaining short columns form a sparser sub-matrix. The denser sub-matrix contains a lot more non-zero elements and fewer columns than the sparser sub-matrix. According to Amdahl’s law [3], the overall performance of the SpMV kernel will be improved if the computation in the denser sub-matrix can be finished efficiently. Now we can tile the denser sub-matrix with texture cache. The non-zero elements are concentrated in a small number of tiles so that we can still gain the benefits from x vector caching as well as avoid the overhead of initializing too many tiles. Here we introduce a threshold parameter to partition the matrix into two sub-matrices. We will discuss how to automatically determine this parameter based on the distribution of the column lengths of the matrix in Section 3.1.3.

Figures 3.1(a) to 3.1(c) illustrate the above transformation procedure on a small sparse matrix. Figure 3.1(a) is the original matrix; Figure 3.1(b) reorders the columns of the matrix in decreasing order of column lengths. In this example, we set the column length threshold to 2. Columns with more than or equal to 2 non-zero elements will be placed in the denser sub-matrix; the other columns with only 1 non-zero element will be placed together in the sparser sub-matrix. Suppose the texture cache
can only hold 2 floating point numbers in this example, the denser sub-matrix with 4 columns will be partitioned into 2 tiles as shown in Figure 3.1(c).

Amongst all the kernels in NVIDIA’s SpMV library, HYB and COO perform best on power-law matrices. The computation in the sparser matrix is run under the HYB kernel, because HYB has the best performance. The computation within each tile of the denser matrix will be performed using COO kernel. The resulting vector $y$ from the denser and sparser sub-matrices will be combined to the final result.

**Observation 3: Performance of COO kernel is limited by thread divergence and serialization.** When computing each tile, the COO kernel cannot utilize the massive thread level parallelism in CUDA efficiently although it is more efficient than the CSR-vector and ELL kernel on such data. In the COO kernel, the inputs are three arrays storing the row indices, the column indices and the values of non-zero elements in the matrix. These three arrays are all divided into equal length intervals. Each interval is assigned to one warp. Note that this partition only equally distributes workload to warps. It does not consider that a row may cross the boundary between two warps. Each warp of threads iterates over an interval in a strided fashion. The stride size equals warp size, and a thread within a warp only works on one element in one stride. A thread first fetches the value in the $x$ vector based on the column index, and then multiplies the $x$ value with the element in matrix $A$ and stores the result in a shared memory space reserved for it. The next step is the *sum reduction* of the multiplication results within one stride by a binary reduction operation. But one stride can contain non-zeros from more than one row. When the reduction operation tries to add two operands, it has to first check whether the two operands are from
the same row in the original matrix. If not, this warp of threads will be serialized due to the thread divergence. This leads to low thread level parallelism in the COO kernel.

**Observation 4: Performances of CSR-vector and ELL kernel are limited by imbalanced workload.** The CSR-vector kernel performs the best when the rows of a matrix are long and with similar length. Non-zeros are stored in row major fashion in CSR format. CSR-vector kernel assigns one warp of threads per row. Each warp iterates on the row with stride size the same as warp size, and performs multiplication and summation operations. After the last iteration on this row, the threads in a warp perform a binary reduction to obtain the final result of this row. In all the summation and reduction operations, the threads do not need to check whether the two operands are from the same row. However, CSR-vector kernel is most efficient when the number of non-zeros in a row is an integer multiple of the warp-size.

The ELL kernel achieves peak performance if there are large number of short rows with similar lengths in the sparse matrix. In the ELL format, all rows have the same length and 0s are padded to the rows shorter than this length. The non-zeros are stored in column major order. A warp of threads is assigned to work on 32 consecutive rows, with each thread working on the multiplication and reduction of one row. The threads within a warp iterate over the columns efficiently with hardware synchronization.

**Observation 5: Tile row lengths follow power-law.** Due to the scale-free property of power-law graphs, we observe that after tiling, the row length within a tile also follows a power-law distribution. We propose to address this fact via a novel storage
Solution 3: Composite tile storage scheme. Our composite storage scheme combines the CSR and ELL formats as follows. Our algorithm starts by ranking the row lengths from high to low. A workload size is defined as the total number of non-zeros in the longest row or several long rows at the top of the ranking, depending on the dataset. We will discuss the auto-tuning of this parameter in Section 3.1.3. Then rows in a tile will be partitioned into approximately balanced workloads. This can be implemented by traversing the row length ranking from top to bottom. A new row is packed into a workload until it exceeds the workload size, then a new workload is initialized. Each workload can be viewed as a rectangle, where the width \( w \) is the length of the first row (the longest row in this workload) and the height \( h \) is the number of rows in this workload. If \( w \geq h \), this workload will be stored in row major in global memory and computed by CSR-vector kernel; otherwise, it will be stored in column major and computed by ELL kernel. Note that if a workload is stored in row major, all rows will be padded to the same length as \( w \) with 0s; and 0s will also be
padded to ensure that $w$ (or $h$) is an integer multiple of warp size when a workload is stored in row (or column) major. After the above partition and transformation of storage format, each workload is assigned to a warp of threads and computed with the most suitable kernel.

The above composite storage scheme is designed for tiles of the denser sub-matrix in the original matrix. However, it can be used for all matrices whose row lengths follow a power-law distribution. We observed that the row lengths in the sparser sub-matrix also follow a power-law distribution. Therefore, we also transform the sparser sub-matrix as one matrix tile into the composite storage format.

Figure 3.2 illustrates how tile 0 from Figure 3.1(c) is transformed in our composite storage scheme on a fictitious architecture with two threads per warp. The rows in tile 0 are first reordered by row length. Suppose we set the workload size to be 4. The first two rows are packed into the first workload, stored in row major and assigned to warp 0 for computation. The two threads in warp 0 first do multiplication and reduction on row 0 using CSR-vector kernel and move to row 1 together. The next two rows are packed together, stored and computed by warp 1 in a similar fashion. The remaining four single element rows are stored in column major and computed by warp 2. The two threads in warp 2 start from the first two rows vertically using ELL kernel and then move to the last two rows.

**Elimination of Partition Camping:** The global memory is divided into 8 equally sized memory partitions of 256-byte width. Concurrent memory requests to the global memory by all the active warps should be distributed uniformly amongst partitions. The term *partition camping* [81] is used to describe the situation when global memory
accesses are congested and queued up at some partitions while the other partitions are idle. All data in strides of 2048 bytes (or 512 floats) map to the same partition.

In our tile-composite format, if the workload size is an integer multiple of 512 floats, then the start memory addresses of all workloads are mapped to the same partition. All warps will queue up at each partition when the warps iterate on their own workload. Thus, we will have the problem of partition camping. To avoid this problem, we add 256 bytes of memory to the end of each workload if the workload size is an integer multiple of 512 floats.

**Sorting Cost:** Sorting is used to re-structure the matrix to improve memory locality. The cost of sorting is relatively cheap when the rows and columns follow power-law. The rows and columns can be bounded by some small number $k$ in the long tail of the power law distribution. These rows or columns can be sorted by counting sort in linear time [33]. The remaining rows or columns can be sorted very quickly. Moreover, we only need to perform the sorting once as a data preprocessing step. In applications such as the power method where the SpMV kernel is called iteratively until the result converges, the cost of sorting can be amortized over the iterations.

### 3.1.2 Multi-GPU SpMV

In this section, we show how our SpMV kernels can scale to large web graphs that cannot fit in the memory of a single GPU. To handle out-of-core matrices, we can either use a single GPU to work on chunks of the matrix in serial, or distribute the chunks to multiple GPUs. Because the single GPU strategy has to move the
data from CPU to GPU in every iteration, the bandwidth of the PCI-Express bus from CPU to GPU (8GB/s) will become the performance bottleneck in the single GPU kernel, because our best kernel can comfortably achieve 40GB/s bandwidth (see Figure 3.3(b)). Hence we devise a method that can use a multi-GPU cluster to compute SpMV kernel on large-scale dataset.

In the cluster, each node keeps one local partition of the matrix. At the end of each iteration, all nodes need to broadcast their local result vector $y$ to the other nodes, so that the other nodes can update their local copy of the vector $x$ for the next iteration. The communication cost of broadcasting, decided by the partition scheme, is the key factor that limits the scalability of the multi-GPU SpMV kernel. Algorithms such as matrix partitioning and graph clustering can be used to minimize this cost. But those algorithms are often more expensive than the iterative SpMV kernel which would be self-defeating. Hence we only consider simple partition schemes such as partition by rows, by columns and by grids.

The communication cost is lower if the matrix is partitioned by rows rather than by columns. Suppose we have $N$ rows and $P$ processors. If the matrix is partitioned by rows, each processor only needs to send out $N/P$ elements of vector $x$. But if partitioned by columns, all processors need to send out $N$ elements. Also partitioning by rows does not necessitate any reduction operations after vector $x$ is gathered. For similar reasons, we can show that partitioning by rows is superior to partitioning by grids. In our multi-GPU SpMV kernel, we choose to partition the matrix by rows with a partition scheme that can assign approximately equal number of rows and equal number of non-zeros in each partition. Such scheme can guarantee both balanced computation workload (number of non-zeros) and balanced communication.
The cost (number of rows) on each node. We use bitonic partitioning [86] for this task. The intuition of bitonic partitioning is as follows: The matrix rows are first sorted by length. Each iteration of the algorithm processes P rows and assigns them to P processors. The processor that got the longest row in the previous iteration will get the shortest row in the current iteration.

Any SpMV kernel can be plugged into this multi-GPU framework to perform local computation. Because power-law graphs are known to be scale-free, we observe that the rows and columns of each partition of a power-law matrix also follow power-law. Hence we can expect our optimized SpMV kernel for power-law matrices to be a good fit for the local computations at each GPU in the cluster.

We next introduce an automatic parameter tuning method to find the optimal parameters for each local kernel, since it is prohibitive to use exhaustive search to find optimal parameter settings for the kernels on every node.

### 3.1.3 Automatic Parameter Tuning

The practical utility of our optimization approach is somewhat limited by the need to carefully tune two parameters – the number of tiles and the partitioning strategy of each tile. We address this limitation as follows.

The first parameter we need to determine is the number of tiles in the composite kernel. It turns out that this parameter is relatively straightforward to estimate. The performance gain of the tiling strategy hinges on the reuse that comes from the temporal locality of the texture cache when accessing vector \( x \). If a matrix column only has one non-zero element, there is no reuse benefit. It turns out that if there is
any re-use, however small, tiling is typically beneficial. Thus, a new tile should not be added if its first column only has a single element (line 7–8 Algorithm 1).

**Algorithm 1** Tile-Composite Kernel Auto-tuning

1: Input: $n$ by $n$ matrix $M$ sorted by column lengths
2: Output: number of tiles and partition size of each tile
3: $TILE\_WIDTH \leftarrow 64K$, $NTile \leftarrow 0$
4: while $NTile < n/TILE\_WIDTH$ do
5:   $StartCol \leftarrow NTILE \times TILE\_WIDTH$
6:   if $M.ColLength[StartCol] \leq 1$ then
7:     break;
8:   else
9:     for $i = StartCol$ to $StartCol + TILE\_WIDTH$ do
10:        $InsertCol(M.Tile[NTile], M.Col[i])$
11:     end for
12:     $WL = Partition(M.Tile[NTile])$
13:     $NTile \leftarrow NTile + 1$
14:   end if
15: end while
16: Return: $NTile, WL$

Next we need to determine how to partition each tile into small rectangular workloads and assign each workload to a warp of threads in the SpMV kernel. Since the tile-composite kernel requires a balanced workload for each warp, the size of one workload directly decides the partition of the tile. The partition problem can be transformed to finding the optimal workload size. Considering the search space of the workload size, the lower bound is the length of the first row in the tile. This is because the rows within each tile are reordered in decreasing order of row lengths, and the longest row cannot be partitioned into two workloads.

The upper bound of the workload size is the total number of non-zeros in the tile divided by the maximum number of active warps available (960 on the Tesla GPU) to fully utilize the GPU resources. Apart from the upper and lower bounds, an additional constraint is that the workload size must be an integral multiple of the first row in this tile (line 11 Algorithm 2). This is because the first workload of this tile
must be a rectangle area of non-zeros where each row is padded to the same length as the first row. These constraints dramatically reduce the search space of viable workload size settings. We must now estimate the performance under each setting and return the one that is predicted to perform the best (Algorithm 2). We next describe our performance model which has broader utility beyond just our use for parameter setting.

**Algorithm 2** Partition($T$): Partition of one tile of matrix

1: **Input:** one tile $T$ from the matrix sorted by row lengths
2: **Output:** optimal workload size to partition $T$
3: $WL_{low} \leftarrow T.RowsLength[0]$ \{Workload lower bound\}
4: $WL_{up} \leftarrow T.NNZ$ \{Workload upper bound\}
5: OptWL $\leftarrow 0$; OptTime $\leftarrow +\infty$; $WL \leftarrow WL_{low}$
6: while $WL \leq WL_{up}$ do
7: \hspace{1em} Time $\leftarrow PM(T,WL)$ \{Performance Modeling of $T$ with WL\}
8: \hspace{1em} if Time $<$ OptTime then
9: \hspace{2em} OptTime $\leftarrow$ Time; OptWL $\leftarrow$ WL
10: \hspace{1em} end if
11: \hspace{1em} $WL \leftarrow WL + T.RowsLength[0]$
12: end while

**Performance Model:** Our model relies on an execution model of CUDA kernels - which forms an offline component in our model - and the non-zero element distribution of the input matrix - which is an online component. The offline component seeks to create a lookup table indexed on the rectangular shape ($w$ columns, $h$ rows) of a workload and its corresponding performance. At runtime the online component given a particular input matrix tile computes an estimate of its runtime cost for different viable workload sizes. The lowest cost option is selected as the partitioning strategy. We next detail both steps.
Given a rectangle workload whose shape is defined by \( w \) and \( h \), we want to construct a lookup table establishing a mapping from the shape of the workload to its performance on one thread warp. We run offline benchmarks to establish this mapping as follows. Each benchmark is for one combination of \( w \) and \( h \), and we artificially construct a matrix in tile-composite format, in which all workloads are set to the same \( w \) by \( h \) shape and there are large number of such workloads to fill the computation pipeline of each streaming multiprocessor and to hide the memory latency. We measure the performance of all realizable combinations of \( w \) and \( h \). This may seem to be an exhaustive process but fortunately we have several constraints that limit the realizable combinations we need to evaluate. Moreover it is a one time offline cost that is independent of the dataset. The first constraint is the upper bound of the workload size. In practice, we can choose a large enough upper bound to cover all possible workloads. The second constraint is that either \( h \) or \( w \) must be a multiple of the warp size (32). Therefore the total number of combinations of \( w \) and \( h \) is relatively small and finite. To reiterate, we only need to construct this mapping once for a given GPU architecture (e.g. Tesla) and this mapping can be repeatedly used for any input data matrix.

In the online component of our model, given a particular input matrix tile we first identify viable workload sizes based on the upper and lower bounds as noted above. Then for a specific workload size we partition the tile into workloads of potentially different shapes but of roughly the same size, in the same way as the tile-composite kernel (Figure 3.2, line 8 – 9 Algorithm 3). We then estimate the performance of the entire tile as the average of the performance of all its constituent workloads as follows.
Algorithm 3 $PM(T, WL)$: Performance Modeling of tile $T$ given workload size

1: **Input**: tile $T$ and workload size $WL$
2: **Output**: Total run time
3: $NWarp = \lceil \frac{T.NNZ}{WL} \rceil$
4: $MAX\_ACT\_WARP \leftarrow MAX\_ACT\_WARP/SM \times NUM\_SM$
5: $I = \lceil \frac{TOTAL\_WARP}{MAX\_ACT\_WARP/SM \times NUM\_SM} \rceil$ \{Number of iterations\}
6: $i \leftarrow 0; j \leftarrow 0$ \{Row Index; Warp Index\}
7: while $i < T.\text{NumRow}$ do
8: \{Partition $T$ with workload $= WL$\}
9: $w_j \leftarrow T.\text{RowLength}[i]; h_j \leftarrow \frac{WL}{w_j}$
10: $\text{Padding}(w_j, h_j, \text{WarpSize})$ \{Padding $w$ or $h$\}
11: $\text{IterId} = \frac{MAX\_ACT\_WARP}{SM}$
12: $P[\text{IterId}] \leftarrow P[\text{IterId}] + \text{Performance}(w_j, h_j)$
13: $\text{Size}[\text{IterId}] \leftarrow \text{Size}[\text{IterId}] + w_j \times h_j$
14: $\text{Cnt}[\text{IterId}] \leftarrow \text{Cnt}[\text{IterId}] + 1$
15: $j \leftarrow j + 1; i \leftarrow i + h_j$
16: end while
17: for $i = 0$ to $I - 1$ do
18: \hspace{1em} $P[i] \leftarrow \frac{P[i]}{\text{Warp}}; t[i] \leftarrow \text{Size}[i]$
19: \hspace{1em} $\text{TotalTime} \leftarrow \text{TotalTime} + t[i]$
20: end for
21: **Return**: $\text{TotalTime}$

After we establish the mapping from $w$ and $h$ to a performance number, we simulate the computation of a matrix tile by looking up performance numbers from the mapping instead of running the actual tile-composite kernel. Here we need to consider the execution model of CUDA hardware. Each streaming multiprocessor (SM) can only serve $MAX\_ACT\_WARP/SM$ number of active warps at a time. So if the total number of warps is larger than the maximum number of active warps on a GPU, the thread warps will be divided into iterations so that each iteration can fit in all the SMs on the GPU. The number of iterations can be computed as (line 5 Algorithm 3): 

$$I = \left\lceil \frac{TOTAL\_WARP}{MAX\_ACT\_WARP/SM \times NUM\_SM} \right\rceil$$ \hspace{1em} (3.1)

The total running time will be the sum of the running time in each iteration (line 19 Algorithm 3): 

$$t = \sum_{i=1}^{I} t_i$$ \hspace{1em} (3.2)
The running time of each iteration $i$ is computed as the total size of the workload in this iteration divided by the average performance of all the warps in this iteration (line 18 Algorithm 3):

$$t_i = \frac{\text{Size}(i)}{P_i} \quad (3.3)$$

where $\text{Size}(i)$ is the total size of all the workloads in iteration $i$ (line 13 Algorithm 3),

$$\text{Size}(i) = \sum_{\text{warp}(j) \in \text{Iter}(i)} w_j \times h_j \quad (3.4)$$

and $P_i$ is estimated by the average performance of all warps in this iteration (line 18 Algorithm 3),

$$P_i = \frac{\sum_{\text{warp}(j) \in \text{Iter}(i)} \text{Performance}(w_j, h_j)}{\text{Total Warps}(i)} \quad (3.5)$$

For given $w_j$ and $h_j$, $\text{Performance}(w_j, h_j)$ can be found in the mapping we established from the offline benchmarks.

Based on Equation 3.1 – 3.5, we can calculate the prediction of the total running time of computing a matrix tile with the given workload size. Our parameter auto-tuning method will call the above performance model as a subroutine to estimate the optimal setting of workload size in each tile (line 7 Algorithm 2).

Note that the thread block size and the number of thread blocks are parameters in all CUDA kernels. In our tile-composite kernel, we assume full occupancy of all multiprocessors, which means there are 32 active warps on each SM at any time. This assumption is reasonable because if there are fewer warps on each SM, the performance will decrease. The maximum size of a thread block is 512 (=16 warps), and each SM can have 8 active thread blocks at maximum. So the 32 active warps on each SM can only be organized as 8 blocks with 4 warps per block, 4 blocks with 8 warps per block, or 2 blocks with 16 warps per block. In practice, the three configurations yield
<table>
<thead>
<tr>
<th>Matrix</th>
<th>Rows</th>
<th>Columns</th>
<th>NNZ</th>
<th>NNZ/Row</th>
<th>Power-law?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>2K</td>
<td>2K</td>
<td>4M</td>
<td>2K</td>
<td>No</td>
</tr>
<tr>
<td>Circuit</td>
<td>171K</td>
<td>171K</td>
<td>0.96M</td>
<td>5.6</td>
<td>No</td>
</tr>
<tr>
<td>FEM/Harbor</td>
<td>47K</td>
<td>47K</td>
<td>2.4M</td>
<td>50.6</td>
<td>No</td>
</tr>
<tr>
<td>LP</td>
<td>4.3K</td>
<td>1M</td>
<td>11M</td>
<td>2632.9</td>
<td>No</td>
</tr>
<tr>
<td>Protein</td>
<td>36K</td>
<td>36K</td>
<td>4M</td>
<td>119.3</td>
<td>No</td>
</tr>
<tr>
<td>Webbase</td>
<td>1M</td>
<td>1M</td>
<td>3M</td>
<td>3.1</td>
<td>Yes</td>
</tr>
<tr>
<td>Flickr</td>
<td>1.7M</td>
<td>1.7M</td>
<td>22.6M</td>
<td>13.2</td>
<td>Yes</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>5.2M</td>
<td>5.2M</td>
<td>77M</td>
<td>14.9</td>
<td>Yes</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>1.9M</td>
<td>1.9M</td>
<td>40M</td>
<td>21.4</td>
<td>Yes</td>
</tr>
<tr>
<td>Youtube</td>
<td>1.1M</td>
<td>1.1M</td>
<td>4.9M</td>
<td>4.3</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.1: Matrix and Graph Datasets

similar performance because the physical executions of the warps are the same on the hardware. In our experiments, we choose 8 warps per block. Since the total number of warps is fixed after we set the workload size, the number of blocks is computed by dividing the total number of warps by 8.

The above auto-tuning and performance modeling method is designed for the tiling part of the tile-composite kernel. A similar method is used to model the sparse part of the matrix. The only difference is that in the sparse part there is no reuse of vector $x$. We only need to run separate offline benchmarks without using the texture cache.

### 3.2 Experiments

In this section, we present experimental results of our optimizations on SpMV kernels and auto-tuning method. We start with describing the datasets used in our evaluation and the configurations of our hardware platform.
Table 3.2: Web Graph Datasets

<table>
<thead>
<tr>
<th>Graph</th>
<th>Nodes</th>
<th>Edges</th>
<th>Edges/Node</th>
<th>Density</th>
<th>Power-law?</th>
</tr>
</thead>
<tbody>
<tr>
<td>it-2004</td>
<td>41,291,594</td>
<td>1,150,725,436</td>
<td>27.9</td>
<td>$6.75 \times 10^{-7}$</td>
<td>Yes</td>
</tr>
<tr>
<td>sk-2005</td>
<td>50,636,154</td>
<td>1,949,412,601</td>
<td>38.5</td>
<td>$7.60 \times 10^{-7}$</td>
<td>Yes</td>
</tr>
<tr>
<td>uk-union</td>
<td>133,633,040</td>
<td>5,507,679,822</td>
<td>41.2</td>
<td>$3.08 \times 10^{-7}$</td>
<td>Yes</td>
</tr>
<tr>
<td>web-2001</td>
<td>118,142,155</td>
<td>1,019,903,190</td>
<td>8.6</td>
<td>$7.31 \times 10^{-8}$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

3.2.1 Dataset and Hardware Detail

Datasets: In our single GPU experiments, we use four web-based graph datasets, including user link relationship graphs from Flickr, LiveJournal and Youtube and a webpage link relationship graph from Wikipedia [74]. All graphs exhibit power-law characteristics. In addition to the graph datasets we also include results on six popular unstructured matrix datasets, representing various scientific kernels, used in previous studies [10]. Among these, one is a 2000 by 2000 dense matrix, which while not sparse, is a useful benchmark to show the maximum bandwidth that each kernel can achieve. Details of these graphs (represented in an adjacency matrix) and matrices are shown in Table 3.1. In the four graph datasets, the number of non-zeros (NNZ) is the number of directed links and the number of rows (or columns) is the number of nodes in the graphs. In our multi-GPU experiments, the web graph datasets used are provided in Table 3.2. These web graphs were crawled using UbiCrawler [17] developed by the members of the Laboratory for Web Algorithmics at the Univerita Degli Studi Di Milano. All of these datasets cannot fit in the memory of one GPU.

Hardware configuration: Our experiments are run in an MPI-based cluster environment. On the CPU side, each node has an Opteron X2 2218 CPU with 8 GB of main memory. On the GPU side, each node is equipped with two NVIDIA Tesla
C1060 GPUs. Each GPU has 30 multiprocessors with 240 processing cores and 4 GB of global memory. The single GPU experiments are run on a single node and a single GPU. The multi-GPU experiments are run on multiple nodes, and each node uses a single GPU. The CPU code is compiled with the gcc compiler version 4.1.2. The GPU code is compiled with CUDA version 3.0.

3.2.2 Single-GPU SpMV Kernel

We compare a CPU implementation of the CSR kernel, all six GPU kernels from NVIDIA’s SpMV library, Baskaran and Bordawekar’s optimized CSR GPU kernel (BSK & BDW) and our two optimized GPU kernels (TILE-COO and TILE-COMPOSITE) on the matrix datasets in Table 3.1. We report the speed of execution in GFLOPS determined by dividing the number of arithmetic operations, which is twice the number of non-zeros in the matrix, by the running time. The running time is averaged over 500 iterations. Since the SpMV kernel is a bandwidth limited problem, we also report the effective bandwidth of each kernel in GB/s, which is the total number of bytes accessed by the kernel divided by the running time. Note that different storage formats have their own data structures. These data structures are counted into the effective memory accesses. All kernels are run in single precision mode. Binding the entire vector \( x \) to texture cache performs consistently better than not binding in all NVIDIA’s kernels [10] and Baskaran and Bordawekar’s kernel [8]. So we only report the performance of these kernels with texture cache binding. We use 256 threads per thread block. This setting is default in NVIDIA’s SpMV library. Under this setting, there are enough warps in each thread block to hide the memory latency and the multiprocessors can be fully utilized. In our tiling method, we have to
set a threshold to decide the number of tiles (the width of each tile is pre-determined by the size of the texture cache and corresponded to 64K columns). In this section, we choose this parameter by exhaustive search to present the best performance that can be achieved by our kernel. Later, we will examine the performance under auto-tuning. The performance of the SpMV kernels on power-law matrices are shown in Figure 3.3. The results on the other matrices are presented in Figure 3.4.

![Performance](image)

**Figure 3.3:** SpMV kernels comparison on matrices representing power-law graphs.

**Performance on power-law matrices:** Our tile-coo and tile-composite methods clearly dominate the other kernels on the Flickr, LiveJournal and Wikipedia datasets. Our tile-composite kernel has an average $1.95x$ speedup over NVIDIA’s best kernel – HYB kernel on these datasets. On Webbase and Youtube datasets,
which are small power-law matrices, COO and HYB kernel perform close to our optimizations. Our Tile-composite kernel is about 13% faster on Webbase and 36% faster on Youtube than the NVIDIA HYB kernel. From Table 3.1, we can see the numbers of rows and columns are low in the Webbase and Youtube matrices, and also the numbers of non-zeros per row and column are low. These properties of the Webbase and Youtube matrices hide the advantages of our optimizations for the following reasons. First, there is little reuse of vector $x$ if non-zeros per column is low. This leads to lesser benefit from our tiling optimization. Second, when the number of columns is small, COO and HYB kernel have better probability of cache hits when they bind the entire vector $x$ to the texture cache. Third, the total number of non-zeros in a tile is low so our composite storage scheme will pad more zeros and thereby cause memory access overhead. We do not report performance of NVIDIA’s PKT kernel on these datasets since the partition step within this kernel does not produce balanced enough packets and leads to kernel failure.

**Performance on Unstructured Matrix Data:** The speed and bandwidth performance of different kernels on non-power-law matrices are shown in Figure 3.4. We immediately observe that our methods while comparing favorably on some of the kernels do not always perform as strongly as the best. In fact on these datasets, interestingly, no single kernel outperforms all others.

Our tiling with composite storage kernel performs the best on the 2000 by 2000 dense matrix with 17.57 GFLOPS speed and 105.5 GB/s bandwidth. This bandwidth utilization is higher than the peak bandwidth of 102 GB/s in the official hardware specification from NVIDIA website. This somewhat surprising result is due to the effect of texture binding of vector $x$ allowing for elements in $x$ to be directly fetched.
from the cache. Our tiling with composite storage kernel runs 30% faster than CSR-vector kernel on the dense matrix. This is because we pad the storage of the matrix in global memory to ensure that all global memory accesses are fully coalesced. The CSR-vector format concatenates all rows together. If one row is not padded to an integer multiple of the warp size, all global memory accesses after this row will not be fully coalesced resulting in a loss in performance.

Baskaran and Bordawekar’s CSR kernel performs best on FEM/Harbor and Protein dataset. Their kernel achieves 12.76 GFLOPS speed and 78.6 GB/s bandwidth on FEM/Harbor, and 15.74 GFLOPS speed and 95.5 GB/s bandwidth on Protein. The bandwidth utilizations are close to the maximum on these two datasets. HYB kernel performs best on the other two non-power-law matrices. It achieves 5.98 GFLOPS
speed and 45.4 GB/s bandwidth on Circuit matrix, and 8.45 GFLOPS speed and 61.6 GB/s bandwidth. On all four non-power-law matrices, our tiling with composite storage kernel is all amongst the top four in speed and bandwidth. Our tiling with composite storage kernel is only 10.5% slower than HYB kernel on Circuit matrix.

The non-zero elements present a relatively balanced distribution in these non-power-law matrices, not as biased as power-law matrices. Our methods first reorder the columns and partition the matrices into a denser and a sparser sub-matrix. This partition will not produce a dense enough matrix in which most of the non-zeros in the original matrix are concentrated. This phenomenon leads to the low performance of our methods on non-power-law matrices, because the tiling of the denser matrix still requires overhead, but does not gain benefit in performance.

**Comparison with CPU SpMV**: Previous works [10, 30] have already illustrated the benefits of GPU vs CPU. Our main point is to demonstrate the benefits of our approach over other GPU-based work on power-law graphs. The CPU results are included for the completeness of our evaluation. We implemented the SpMV kernel with CSR format on the CPU. CSR format is the most efficient on CPU among different sparse matrix formats. We ran experiments with the CPU kernel on all datasets in Table 3.1. The GPU kernels significantly outperform the CPU kernel in almost all settings. GPU CSR kernel is the slowest kernel on GPU. It is slower than CPU kernel on the Dense matrix data because the clock rate of one GPU processor is lower than CPU. The GPU kernels perform dominantly faster than CPU kernel in all the other formats with speedups ranging from 2.05x to 37.31x.
3.2.3 Graph Mining Applications

In this section we discuss how the single GPU performance of our SpMV kernel translates to performance on important graph mining algorithms. A large class of graph mining algorithms leverage the SpMV kernel iteratively to perform computation until the algorithms converge, e.g. PageRank [21, 83], HITS [61] and Random Walk with Restart [85, 107]. These algorithms first transform the adjacency matrix of a graph and then operate on the transformed matrix. The graph dataset used by these algorithms usually have strong power-law properties, hence the number of non-zeros on each row or column of the corresponding matrix will follow a power-law distribution. The skew of the distribution leads to poor load balancing and low memory access efficiency on GPU.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures}
\caption{Performance and bandwidth of PageRank.}
\end{figure}

**PageRank**: We run Equation 2.1 iteratively with the SpMV kernels and check whether $p$ converges on the 4 graph datasets in Table 3.1. The speed and bandwidth performance of PageRank based on the four kernels are shown in Figure 3.5(a) and
Figure 3.5(b). The total running time on each graph is shown in Table 3.3 in comparison to a CPU implementation of PageRank. Our tile-coo and tile-composite kernel achieve about 2x speedup over COO and HYB kernel on Flickr, LiveJournal and Wikipedia graphs. Our optimizations are marginally better than NVIDIA’s COO and HYB kernel on the Youtube graph (reasons noted earlier). Compared with the CPU PageRank, all GPU implementations achieve between 18x and 32x speedup.

HITS:

In each iteration of HITS algorithm, a 2|V| by 2|V| matrix in Equation 2.3 is multiplied by a vector combined with $\vec{a}$ and $\vec{h}$. Then the first and second half of the resulting vector are normalized to sum to 1 separately. Each normalization requires a reduction operation on the vector and a division of the vector by a constant. A convergence check is also needed at the end of each iteration. Each iteration of our HITS implementation involves one SpMV kernel, three parallel reduction kernels (two for normalization and one for convergence check) and two vector division by constant kernels. The vector division by constant kernel can be implemented very efficiently in the same way as vector addition. On our implementation of the HITS algorithm we compare the performance of our four GPU SpMV kernels on the four graph datasets. The speed and bandwidth performance are shown in Figure 3.6(a) and Figure 3.6(b).

<table>
<thead>
<tr>
<th>Graph</th>
<th>CPU</th>
<th>COO</th>
<th>HYB</th>
<th>TILE-COO</th>
<th>TILE-Comp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr</td>
<td>23.99</td>
<td>1.67</td>
<td>1.60</td>
<td>0.90</td>
<td>0.83</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>82.23</td>
<td>6.19</td>
<td>5.57</td>
<td>3.75</td>
<td>3.44</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>52.12</td>
<td>2.99</td>
<td>2.83</td>
<td>1.76</td>
<td>1.63</td>
</tr>
<tr>
<td>Youtube</td>
<td>11.81</td>
<td>0.72</td>
<td>0.66</td>
<td>0.68</td>
<td>0.65</td>
</tr>
</tbody>
</table>

Table 3.3: Running time of PageRank (in seconds)
Our TILE-COO and TILE-Composite kernels perform better than COO and HYB kernels in all four datasets. On Flickr, LiveJournal and Wikipedia, the speedups are similar to those observed in PageRank algorithm. On Youtube, our optimizations are actually a bit faster when compared to the NVIDIA kernels in spite of the relatively small size of the dataset. Combining the two matrices into one in the HITS algorithm results in a larger and sparser matrix making it more amenable to our optimizations.

The total running time compared with CPU implementation is listed in Table 3.4. We observe a 17x to 29x speedup of the GPU implementations over the corresponding CPU implementation.

![HITS Performance](image1)

(a) HITS Performance

![HITS Bandwidth](image2)

(b) HITS Bandwidth

Figure 3.6: Performance and bandwidth of HITS.

**Random Walk with Restart:**

In each iteration of Random Walk with Restart algorithm, there is a matrix-vector multiplication followed by a vector addition and a convergence checking operation shown in Equation 2.4. In our implementation, we use the GPU SpMV kernels for matrix-vector multiplication, and GPU parallel reduction for checking convergence in the same way as PageRank. An efficient vector addition kernel is also implemented.
by assigning one GPU thread to compute one element in the resulting vector. Note that RWR is an interactive application, we randomly select 25 query nodes in our experiment and the performance is reported by averaging (arithmetic mean) the result of each query. The number of computations per iteration is the same no matter which node is selected as query, so the experiment results of the randomly selected 25 query nodes can reflect the speed of different SpMV kernels. Since RWR operates on undirected graphs, we treat each link in our directed graph datasets as an undirected link in our experiments. The speed and bandwidth performance of RWR implementations on four graph datasets based on four GPU SpMV kernels are shown in Figure 3.7(a) and Figure 3.7(b). The total running time is listed in Table 3.5.

We observe similar performance results as in the case of PageRank. Our optimized TILE-COO and Tile-Composite kernels are 1.5x to 2.0x as fast as COO and HYB kernels on Flickr, LiveJournal and Wikipedia graphs. The four kernels perform about the same on Youtube graph. All GPU implementations are 13x to 37x faster than CPU implementation. The best speedup is achieved by our TILE-Composite kernel on Wikipedia graph.

<table>
<thead>
<tr>
<th>Graph</th>
<th>CPU</th>
<th>COO</th>
<th>HYB</th>
<th>TILE-COO</th>
<th>TILE-Comp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr</td>
<td>4.97</td>
<td>0.40</td>
<td>0.38</td>
<td>0.23</td>
<td>0.21</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>44.88</td>
<td>3.82</td>
<td>3.33</td>
<td>2.41</td>
<td>2.24</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>39.36</td>
<td>2.73</td>
<td>2.45</td>
<td>1.52</td>
<td>1.37</td>
</tr>
<tr>
<td>Youtube</td>
<td>4.35</td>
<td>0.33</td>
<td>0.30</td>
<td>0.26</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 3.4: Running time of HITS (in seconds)
3.2.4 Multi-GPU PageRank on Web Graphs

Figure 3.8 shows the performance results of computing PageRank with the multi-GPU framework introduced in Section 3.1.2 on the four web graph datasets in Table 3.2. The solid lines show the performance of using the Tile-Composite kernel; the dotted lines indicate the performance of using NVIDIA’s HYB kernel. The lines for the sk-2005 and uk-union datasets start from 3 and 6 GPUs, because these datasets are very large and can only fit in the memory of at least 3 and 6 GPUs respectively. From the figures, we observe that our multi-GPU framework and partition scheme can comfortably handle web graphs with billions of edges. For example on 10 GPUs, the distributed implementation of PageRank algorithm with our multi-GPU framework can achieve about 23GFLOPS performance with 70% parallel efficiency (on sk-2005) with the Tile-Composite kernel. On the two small datasets, it-2004 and web-2001, our Tile-Composite kernel achieves about 80% parallel efficiency with 4 GPUs and 60% parallel efficiency with 6 GPUs. All curves tend to flatten out after a point. This is because the workload size per GPU is low and the communication
Table 3.5: Average running time of Random Walk with Restart (in seconds)

<table>
<thead>
<tr>
<th>Graph</th>
<th>CPU</th>
<th>COO</th>
<th>HYB</th>
<th>TILE-COO</th>
<th>TILE-Comp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr</td>
<td>8.25</td>
<td>0.59</td>
<td>0.56</td>
<td>0.33</td>
<td>0.29</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>36.99</td>
<td>2.85</td>
<td>2.60</td>
<td>1.73</td>
<td>1.52</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>23.23</td>
<td>1.46</td>
<td>1.35</td>
<td>0.71</td>
<td>0.62</td>
</tr>
<tr>
<td>Youtube</td>
<td>2.32</td>
<td>0.14</td>
<td>0.13</td>
<td>0.14</td>
<td>0.13</td>
</tr>
</tbody>
</table>

overheads begin to dominate and limit speedup. We should also emphasize that the performance of Tile-Composite kernel is about $1.55\times$ faster than HYB kernel on all datasets.

3.2.5 Parameter Auto-tuning

Next we will present experimental results to validate the auto-tuning method and the performance model introduced in Section 3.1.3. In these experiments we use the five matrices representing power-law graphs in Table 3.1. In our offline benchmarks, we set the upper bound of the workload size to 32768. This is a very conservative upper bound and should suffice for most practical matrices that fit on the Tesla architecture. This number is sufficiently large for both the tiling and the sparse parts of all the matrix datasets, because there will be at least 960 warps (full occupancy) in each kernel, which correspond to about 31M non-zero entries in each tile.

In the first experiment we compare our heuristic approach for determining the number of tiles in our composite strategy. Figure 3.9 shows the number of tiles from the exhaustive search and the auto-tuning method. On the Webbase and Wikipedia matrices, our auto-tuned parameters are exactly the same as exhaustive searched results. We can see that our predicted number of tiles are very close to the optimal numbers on the other three datasets.
Figure 3.8: Scalability of multi-GPU PageRank on web graphs

Figure 3.10 presents the optimal performance number by exhaustive search versus the performance number produced by using the auto-tuned number of tiles and partitioning strategies. The blue bars represent the optimal performance of exhaustive search; the yellow bars represent the results by running the tile-composite kernel with the auto-tuned parameters. On the Webbase and the Wikipedia matrices, the auto-tuning method achieves optimal performance. On the other datasets, the auto-tuned performance is within 3% of the optimal performance which is an excellent result.

For our kernel auto-tuning method we only need the performance model to predict the relative performance trend under different parameter settings so as to automatically select the optimal parameter. However we also want to evaluate how accurately our performance model can predict the absolute performance of the tile-composite kernel. Figure 3.11 presents the measured and the predicted performance both using the number of tiles and the partition size parameters produced by our auto-tuning method. The blue bars are the results by running the kernel on the GPU, while the yellow bars are the simulation results using our performance model. We can see the predictions are accurate, and they are all within roughly 20% of the measured results.
The error is largely attributable to the fact that we use the average performance of the different warps on the streaming multiprocessor to estimate the overall performance, and the fact that the lookup table relies on synthetic benchmarks in which all workloads are of the same shape. To reiterate we should note that this error in prediction of actual performance does not significantly impact our auto-tuning method since there relative performance is what matters.

![Figure 3.9: Auto-tuning: Auto vs Exhaustive searched number of tiles.](image)

### 3.3 Discussion

The power-law property is commonly observed in the datasets of graph mining problems. Our experimental results show that the absolute performance of SpMV kernel on power-law matrices is much lower than non-power-law matrices. Our work is intended to find a better representation of matrices on GPU that is suitable for power-law graphs. Our extensive experiments have demonstrated the effectiveness of
Figure 3.10: Auto-tuning: Auto vs Exhaustive searched performance.

Figure 3.11: Performance modeling: Predicted vs Measured performance.
our optimizations on matrices representing power-law graphs. To broaden the scope of our study, we discuss the applicability of our optimization techniques on general matrices.

**Tiling:** Our partially tiling optimization employs a greedy heuristic. The benefit of tiling comes from the fast accesses of the vector \( x \) from the texture cache. The cost of each tile is the random writes to the resulting vector \( y \). Both the benefit and the cost increase with the number of non-zeros in a tile. But the cost of random writes is bounded by the length of the vector \( y \). After the cost of a tile reaches this bound, the more non-zero elements, the more benefit a tile can gain. Our tiling optimization starts with the densest columns, which is the most beneficial, and greedily finds the following tiles. If the non-zeros of a matrix concentrate in the first few tiles, our partially tiling optimization can stop after finishing the majority of computation work without paying extra cost for the sparse columns. Power-law matrices are a subset of matrices that satisfy this property. Our experimental results are consistent with the above discussion. The only difference between COO and tile-coo kernel is tiling. On power-law matrices, tile-coo kernel performs consistently better than COO (Figure 3.3). On non-power-law matrices, tile-coo kernel is better than COO, but the benefit is very marginal (Figure 3.4).

**Composite Storage:** Our composite storage optimization consists of two parts: combination of CSR and ELL storage, and padding workload to warp size. The tile-composite kernel performs better than tile-coo kernel on both power-law and non-power-law matrices. This storage scheme can be applied to general matrices, although memory overhead of padded zeros should be considered as a constraint.
**Performance Modeling:** Our performance model improves the practical utility of the proposed optimizations. We only need to build the performance model once for the same hardware. The model does not rely on the power-law property of the matrix. Given an arbitrary matrix, we can get a relatively accurate prediction of the performance of our tile-composite kernel from the performance model before conducting large-scale experiments. More importantly, the CSR, CSR-vector and ELL kernels from NVIDIA can be modeled as special cases of our tile-composite kernel under the framework of our performance model. The CSR and CSR-vector kernel can be treated as the tile-composite kernel with a single tile and only CSR storage in the composite storage scheme; The ELL kernel can be seen as the tile-composite kernel with a single tile and with only ELL storage. With the generality of our performance model, the performance of different kernels can be predicted by plugging in the data to the model first. The best predicted kernel can be chosen to perform real computation of the data.

### 3.4 Conclusions

In this chapter, we proposed architecture conscious optimizations for the sparse matrix-vector multiply kernel on GPUs and studied the implications of this effort for graph mining algorithms. Our optimizations take into account both the architecture features of GPUs and the characteristics of graph mining applications. Our tiling approach utilizes the texture cache on GPUs in a more efficient way than previous work and provides much better memory locality. Our tiling with composite representation leverages the power-law characteristics of large graphs in graph mining problems. We have obtained significant performance improvement over the state-of-the-art on such
graph based matrix datasets. We also present empirical evaluations of applying our optimizations to PageRank, Random Walk with Restart and HITS algorithms. On these algorithms, our best kernel is 1.8 to 2.1 times faster than an industrial strength GPU competitor and from 18 to 32 times faster than a similar CPU implementation. The high performance of our optimizations relies on carefully tuning of parameters. We proposed a performance model of SpMV kernel and use the model to automatically tune our tile-composite kernel. We further extend our optimizations to handle web-scale graph datasets on an MPI-based cluster.
Chapter 4: Compact Storage of Social Network Content and Topology

The firehose of data generated by users on social networking and microblogging sites such as Facebook and Twitter is enormous. The data can be classified into two categories: the textual content written by the users and the topological structure of the connections among users. Real-time analytics on such data is challenging with most current efforts largely focusing on the efficient querying and retrieval of data produced recently. In this chapter, we present a dynamic pattern driven approach to summarize social network content and topology. We develop a novel approach to maintain an in-memory summary while retaining sufficient information to facilitate a range of user-specific and topic-specific temporal analytics. We empirically compare our approach with several state-of-the-art pattern summarization approaches along the axes of storage cost, query accuracy, and efficiency using real data from Twitter. We find that the proposed approach is not only scalable but also outperforms existing approaches by a large margin.

4.1 Introduction

Microblogging, a lightweight and easy form of communication within social networks such as Facebook, Google+ and Twitter, has become ubiquitous in its use
with over 4 billion mobile devices worldwide of which over 1 billion support smart services. An increasing number of organizations and agencies are turning to extract and analyze useful nuggets of information from such services to aid in functions as diverse as emergency response, viral marketing, disease outbreaks, and predicting movie box office success. A fundamental challenge for effective human-computer interaction (querying and analytics) is the scale of the data involved. Twitter for instance has over 200 million users (and growing) and several hundred million tweets per day. Supporting interactive querying and analytics requires novel approaches for summarizing and storing such data.

Given the diverse nature of applications, a number of queries may be of interest. Queries such as: What are the currently trending topics?; What did a specific user tweet about today or yesterday? are straightforward to support since one only needs to maintain recent data to answer such queries. However, often times users and organizations are interested in capturing high level trends about both current and past activity – particularly highly trending past activity to understand the evolution of user interests and topic trending. For example complex queries of the following form would be of interest: What topics were people talking about in a specific time interval (2 weeks in the past)?; How has a particular topic evolved across multiple time intervals?; How have a user’s or a group’s tweets, or topics they tweet on changed over time? Answers to such questions may enable organizations to understand questions related to the lineage of topic evolution as well as to better understand user interests, their influence, and possibly build a model of trust for specific users and groups.

Answering such queries in real-time is challenging simply because of the scale of the data that is produced – the memory footprint will grow linearly with time and it
can easily overwhelm the capacity of even the most powerful computer systems. The data generated by social networking services can be classified into two categories: the textual content written by the users (e.g. tweets in Twitter) and the link structure of user connections (e.g. follower – followee relationship in Twitter). The textual content carries the information that people want to share with their friends. It is large-scale and streaming in nature because with hand-held smart phones and tablets, people can write new posts almost anywhere at any time. The link structure captures how the textual content will spread through the social network of users. The user connections are relatively stable compared with the high speed content stream, but it is also in large-scale because of the large number of active users on modern social networks such as Facebook and Twitter.

In this study, we aim to build a summary of data generated by modern social networks, focusing on messages and user links, which can fit in a limited memory budget and can help to answer complex queries. In our view the desiderata for such a framework include: 1) efficient, incremental summary construction (ideally using a single pass and at pace with data influx rate); 2) budgeted memory which grows at most logarithmically with data influx; and 3) support for complex querying and analytics with low reconstruction error (particularly on more recent data, or on highly trending data). Ideally, we would like to be able to answer queries about network content such as the topics in a time interval in the past, evolutionary events related to specific topics across multiple time intervals, and network topology queries about social connections of one or a group of users.

The elements of our approach include: a) SPUR, a batch summarization and compression algorithm that relies on a novel notion of pattern utility and ranking
which can be incrementally updated; b) D-SPUR, a dynamic variant of SPUR that accordingly merges summaries and maintains pyramidal time frames that grows logarithmically while enabling querying at multiple temporal contexts; c) G-SPUR, a lossless compression algorithm for graph data to reduce the storage space of user link structures.

We compare the effectiveness of SPUR variants against state-of-the-art pattern summarization algorithms on a large corpus of Twitter data along the axes of compressibility, reconstruction error, efficiency and flexibility in querying. We find that the SPUR variants are up to two orders of magnitude faster and can produce summaries with much lower reconstruction errors than extant approaches. Furthermore, maintaining temporal information in D-SPUR enables the approximate reconstruction of original data over arbitrary time intervals facilitating novel complex queries. We also demonstrate the efficacy of G-SPUR in speeding up link structure mining kernels.

Figure 4.1: Summarization via Pattern Utility and Ranking (SPUR) Framework

4.2 Stream Summarization of Network Content

In this section, we introduce our method of summarizing the user generated content from a social network. The network content is in the form of a high speed
message stream fluxing into the data processing system. We propose a novel stream processing framework (Figure 4.1) to summarize the input stream with efficient, incremental summary construction and budgeted memory footprint. Given the input message stream with proper word stemming and stop-word removal performed, we divide it into approximately equal-sized batches, e.g. one hour per batch (the first arrow in Figure 4.2). To compress each batch of messages into a summary object which can fit in a constant memory budget $M$ (the second arrow in Figure 4.2), we describe our SPUR algorithm in Section 4.2.1. Then in Section 4.2.2 we discuss the D-SPUR algorithm which ensures the summary size grows logarithmically with time.

Figure 4.2: Division and compression of message stream

### 4.2.1 SPUR

We develop a novel algorithm called SPUR (Summarization via Pattern Utility and Ranking) to summarize a batch of transactions with low compression ratio and high quality in a highly scalable fashion. Our basic idea of compressing a batch of tweets is to replace individual words with frequently used phrases that can cover the original content of the tweets. Consider the example in Figure 4.3 (left), each column represents a word and each row represents a tweet. The original tweets need 24 words to be stored in memory. However, if we use the frequent phrases as patterns to represent the tweets, we can save 10 of 24 words (Figure 4.3 right). Our approach
represents each tweet as a transaction of words and a batch as a set of transactions. We cast the challenge of finding frequent phrases as a frequent itemset mining problem. To compress a batch of tweets into a summary with memory budget $M$, we aim to reduce the storage size by covering the transactions with frequent patterns. There are three main challenges one need to address: compressibility, scalability and quality (of compressed summary).

Algorithm 4 provides the pseudo code of SPUR. The algorithm receives a batch of transactions $B$, a memory budget $M$, a support threshold $\sigma$ and a false positive rate $f$ as input. It outputs a summary which can fit in the memory budget $M$ with false positive rate lower than $f$. First, our algorithm mines frequent itemsets above the support threshold $\sigma$ as candidate patterns (line 1). We use LCM [108] for our purposes. Second, we define a function to capture the utility of each pattern in terms of compressing the transactions (line 3). Third, we rank all the candidates by their utility values and insert them into a priority queue (lines 2 – 4). Finally, we iteratively

![Figure 4.3: A batch of tweets compressed to a summary](image)
select the top ranked pattern to cover the items in transactions until we reach the
memory budget $M$ or the top pattern is not cost effective for compression any more
(lines 5 – 13).

**Algorithm 4 SPUR($B$, $M$, $\sigma$, $f$)**

1: $P \leftarrow$ MineFrequentPatterns($B$, $\sigma$); $Q \leftarrow \phi$; $\text{size} \leftarrow 0$;
2: for all $p \in P$ do
3: 
4: \hspace{1em} $p$.utility = Utility($p$, $f$); $Q$.insert($p$);
5: end for
6: while $\text{size} < M$ do
7: \hspace{1em} $p \leftarrow Q$.top;
8: \hspace{2em} if $p$.utility $\geq 0$ then
9: \hspace{3em} Replace($B$, $p$); \{Replace items using $p$\}
10: \hspace{2em} UpdateRank($Q$, $p$); \text{size} = \text{size} + p$.cost;
11: \hspace{1em} else
12: \hspace{2em} break;
13: end if
14: end while

A coverage of transaction $T_i \in B$ using pattern $p$ will replace the items in $T_i \cap p$
with a pointer to $p$. Two types of errors will be introduced: false negative errors are
the items in the original data but are not covered by any pattern; false positive errors
are items not belonging to a transaction but are introduced by a pattern coverage, i.e.
the items in $p \setminus T_i$. False negative/positive rate is the ratio of false negative/positive
errors over the total number of items in the transactions. In Algorithm 4, $f$ is a
threshold to control false positive rate; and the false negative rate is controlled by $\sigma$
and $M$ together where $\sigma$ decides the infrequent items dropped by frequent pattern
mining and $M$ controls the selection of frequent items in the summary, thus indirectly
determines the frequent items that will be dropped. Next, we will first introduce the
definition of pattern utility.

**Pattern Utility and Ranking:** We define the utility of a pattern $p$ covering a
transaction $T_i$ as:
\[ u(p, T_i) = |T_i \cap p| - 1 - |p \setminus T_i| \quad (4.1) \]

where \(|T_i \cap p|\) captures the storage saved by \(p\), 1 is the storage cost of a pointer to \(p\) in the compressed representation of \(T_i\) and \(|p \setminus T_i|\) is the amount of false positive errors. Here \(|T_i \cap p| - 1\) records the total savings in storage space whereas \(|p \setminus T_i|\) penalizes false positive errors \(^6\).

There is also a cost of space for storing pattern \(p\), because we need to record the actual items in \(p\). But this is not a cost for a coverage with an individual transaction, but the cost for a set of transactions. Given a set of transactions \(C \subseteq B\), the total utility of covering all transactions in \(C\) with \(p\) is defined as the sum of the utilities on all transactions in \(C\) less the cost of storing pattern \(p\):

\[ U(p, C) = \sum_{T_i \in C} u(p, T_i) - |p| \quad (4.2) \]

The compression utility \(Utility(p)\) of pattern \(p\) is the maximum value of \(U(p, C)\) among all \(C \subseteq B\). The coverage set \(C(p)\) of \(p\) is defined as the set of transactions that yields this maximum value. So,

\[ Utility(p) = \max_{C \subseteq B} U(p, C) \quad C(p) = \arg \max_{C \subseteq B} U(p, C) \quad (4.3) \]

For example, in Figure 4.3, the best compression using pattern \(p_1\) is to cover transactions \(T_1, T_2\) and \(T_3\). So \(C(p_1) = \{T_1, T_2, T_3\}\), \(u(p_1, T_i) = |T_i \cap p_1| - 1 - |p_1 \setminus T_i| = 3\) for \(i = 1, 2, 3\) and \(Utility(p_1) = \sum_{i=1}^3 u(p_1, T_i) - |p_1| = 5\). Our compression algorithm needs to find the value of \(Utility(p)\) for each pattern \(p\) and the transactions.

\(^6\)A pattern coverage will not introduce false negative errors, because the support threshold in frequent pattern mining decides the infrequent items we dropped.
in \( C(p) \) with the constraint that the false positive rate should be below a threshold \( f \).

Let’s first consider a simple case where no false positive errors are allowed in a coverage. In this case \( p \) can only cover transactions that contain \( p \). Suppose pattern \( p \) with support \( \sigma(p) \) and length \( l(p) \), if we use \( sup(p) = \{ T \mid p \subseteq T \} \) to represent the set of all transactions that contain \( p \), then \( C(p) = sup(p) \). Therefore,

\[
Utility_{no fp}(p) = \sum_{T_i \in sup(p)} (|T_i \cap p| - 1 - |p \setminus T_i|) - |p|
\]

\[
= \sum_{T_i \in sup(p)} (l(p) - 1) - l(p)
\]

\[
= l(p) \cdot \sigma(p) - \sigma(p) - l(p)
\]

Next, let’s increase the complexity of the pattern coverage problem by allowing false positive errors but no limitation of the false positive rate. Each transaction \( T_i \) will add \( u(p, T_i) \) to \( Utility(p) \). So the value of \( Utility(p) \) is at maximum when all \( T_i \)'s with \( u(p, T_i) \geq 0 \) are in \( p \)'s coverage set \( C(p) \). We can rewrite the definition of \( u(p, T_i) \) as:

\[
u(p, T_i) = |T_i \cap p| - 1 - (|p| - |T_i \cap p|)
\]

\[
= 2 \cdot |T_i \cap p| - 1 - l(p)
\]

So the set of transactions that can maximize \( p \)'s utility is \( C(p) = \{ T \ s.t. |T \cap p| \geq (l(p) + 1)/2 \} \). It is inefficient if we intersect all possible pattern and transaction pairs. The following theorem provides a faster way to find \( C(p) \) by only considering \( p \) and its sub-patterns’ supporting transactions.
Theorem 1 Suppose \( C'(p) = \{ \bigcup \text{sup}(p_i) \mid p_i \subseteq p \text{ and } l(p_i) \geq (l(p) + 1)/2 \} \), where \( \text{sup}(p_i) \) represents all transactions containing \( p_i \), then \( C(p) = C'(p) \).

Proof 4.2.1 First show \( C(p) \subseteq C'(p) \).

\[ \forall T_i \in C(p), |T_i \cap p| \geq (l(p) + 1)/2. \text{ Let } p' = T_i \cap p, \text{ then } p' \subseteq p \text{ and } l(p') \geq (l(p) + 1)/2. p' \subseteq T_i \Rightarrow T_i \in \text{sup}(p') \subseteq C'(p). \text{ So } C(p) \subseteq C'(p). \]

Then show \( C'(p) \subseteq C(p) \).

\[ \forall T_i \in C'(p), \exists p_i \text{ s.t. } T_i \in \text{sup}(p_i), p_i \subseteq p \text{ and } |p_i| \geq (l(p) + 1)/2. \text{ Then } p_i \subseteq T_i \cap p \text{ and } |T_i \cap p| \geq |p_i| \geq (l(p) + 1)/2 \Rightarrow T_i \in C(p). \text{ So } C'(p) \subseteq C(p). \]

Theorem 1 shows that the utility value of a pattern \( p \) can be maximized by covering all transactions that contain sub-patterns of \( p \) with more than half of the length of \( p \). We essentially replace \( p \)'s sub-patterns with \( p \) in those transactions. This process will reduce the storage size by introducing false positive items. However, we cannot control the upper bound of the false positive rate in this strategy. Next, we will discuss how we can guarantee that the false positive rate in the summary is below a threshold \( f \).

Suppose \( p_i \) is a sub-pattern of \( p \), if we replace \( p_i \) with \( p \) in the transactions that contain \( p_i \) but not \( p \), the false positive rate is \( 1 - l(p_i)/l(p) \). Therefore, longer sub-patterns will introduce lower false positive rate. To control the false positive rate below a threshold \( f \), Algorithm 5 first sorts the sub-patterns of \( p \) from long to short (line 3–4) and keeps replacing the sub-patterns in this order until the false positive rate is higher than \( f \) (line 5–21). With this greedy strategy, Algorithm 5 can find the maximum utility of a pattern below a false positive rate threshold. Algorithm 5 also generates the transactions in the coverage (\( p\_coverage\_set \)) as well as the sub-patterns.
Algorithm 5 Utility($p, f$)

```plaintext
1: $p\.utility \leftarrow l(p) - \sigma(p) - \sigma(p) - l(p); p\.coverage_set \leftarrow sup(p);
2: $p\.replaced_patterns \leftarrow \{p\}; area \leftarrow l(p) \cdot \sigma(p); f\_error \leftarrow 0; \{\text{Safe to add transactions without false positive}\}$
3: $sub(p) \leftarrow \text{all sub-patterns of } p;$
4: sort $sub(p)$ by pattern length from long to short;
5: for all $p_i \in sub(p)$ do
6: if $|p_i| \geq (|p| + 1)/2$ then
7: transactions $\leftarrow sup(p_i) \setminus p\.coverage_set; \{\text{First find the new transactions that can potentially be covered.}\}$
8: new area $\leftarrow area + l(p) \cdot |\text{transactions}|$;
9: new error $\leftarrow f\_error + (l(p) - l(p_i)) \cdot |\text{transactions}|$;
10: if new error / new area $\leq f$ then
11: $p\.utility \leftarrow p\.utility + |\text{transactions}| \cdot (2 \cdot l(p_i) - l(p) - 1); \{\text{Update } p\text{'s utility}\}$
12: $p\.coverage_set \leftarrow p\.coverage_set \cup \text{transactions};$
13: $p\.replaced_patterns \leftarrow p\.replaced_patterns \cup \{p_i\};$
14: area $\leftarrow \text{new area}; f\_error \leftarrow \text{new error};$
15: else
16: break;
17: end if
18: else
19: break;
20: end if
21: end for
```

that are replaced by $p$ ($p\.replaced_patterns$). Note: we define the utility of singleton patterns as 0. Instead of making a new pattern with only a single item and storing pointers to it, we can directly store the item id in each transaction and it will cost the same amount of memory as the original data.

The SPUR algorithm calls Algorithm 5 to initialize the ranking of pattern utilities. However, the ranking can change dynamically during the compression iterations. Existing approaches either use a static approximation (e.g. Krimp [103]) or if dynamic, need multiple passes of the data. Hence, they do not fit in the setting of summarizing data streams. Next, we will show how our utility function can be dynamically and efficiently updated without accessing the original data.

Compression with Dynamic Ranking Adjustment

There are three categories of patterns whose utility values will be affected by a top 1 pattern $p$: $p$’s super-patterns, sub-patterns and overlapping patterns (Figure 4.4).
Algorithm 6 elaborates how we penalize the utilities of these three types of patterns when \( p \) is selected. For an affected pattern \( p_a \), we first find the transactions in the intersection of \( p_a \)'s and \( p \)'s coverage sets (line 2). The effective coverage area of \( p_a \) will be changed by including \( p \) in these transactions, because the common items of \( p \) and \( p_a \) are already covered by \( p \). Based on the type of \( p_a \), we can penalize its utility value by the total area covered by \( p \) already. For a super-pattern \( p_{super} \), the area covered by \( p \) is \( l(p) \) per transaction (line 4) and \( l(p_{overlap} \cap p) \) for a overlapping pattern \( p_{overlap} \) (line 12). For a sub-pattern \( p_{sub} \), the area covered by \( p \) is \( l(p_{sub}) \) per transaction. But since the transactions covered by \( p \) are not in \( p_{sub} \)'s coverage set any more, the space of the pointers to \( p_{sub} \) in those transactions are saved. So each transaction is penalized by \( l(p_{sub}) - 1 \) (line 6). We can see our algorithm only needs to deduct a value of area from a pattern's utility without scanning the items in the transactions.

There are several implementation and performance related issues worth mentioned: First, our algorithm needs to find the sub-/super-/overlapping-pattern relationships among all patterns. However, once these relationships are established, they
Algorithm 6 UpdateRank(Q, p)

1: for all \( p_a \in p \)'s sub-/super-/overlapping set do
2:   covered_set ← \( p_a \).coverage_set \( \cap \) p.coverage_set;
3:   if \( p_a \) is \( p \)'s super-pattern then
4:      \( p_a \).utility ← \( p_a \).utility - \( l(p) \cdot |\text{covered}\_\text{set}|;\)
5:   else if \( p_a \) is \( p \)'s sub-pattern then
6:      \( p_a \).utility ← \( p_a \).utility - \((l(p_a) - 1) \cdot |\text{covered}\_\text{set}|;\)
7:      \( p_a \).coverage_set ← \( p_a \).coverage_set \( \setminus \) \text{covered}\_\text{set};
8:   if \( p_a \).coverage_set is empty then
9:      Q.remove(\( p_a \)); {\( p_a \) has been replaced by \( p \).}
10: end if
11: else
12:      \( p_a \).utility ← \( p_a \).utility - \( l(p_a \cap p) \cdot |\text{covered}\_\text{set}|;\)
13: end if
14: end for

are reused by the many iterations of the SPUR algorithm. Second, the overlapping-patterns with \( p \) are found by taking the union of all super-patterns of \( p \)'s sub-patterns. Third, we use a max heap to maintain the candidate pattern queue dynamically.

4.2.2 D-SPUR

We now present D-SPUR, the dynamic version of SPUR. In D-SPUR, we enhance and modify the pyramidal time window suggested by Aggarwal et al. [?] for clustering in data streams. Our enhancements center on the fact that we need to find an effective way to manage the stream of summary objects produced by SPUR while limiting the growth of memory footprint and reconstruction error (especially on recent and trending data). D-SPUR summarizes dynamic message streams by maintaining the pyramidal time window in Figure 4.5.

The input to the pyramidal time window is a stream of summary objects, each with memory size \( M \). (right half of Figure 4.2). A level of the time window can hold two summary objects. Figure 4.5 demonstrates how the summary objects are inserted into the time window. Initially, the time window is empty, so Summary 1 and 2 can be directly inserted into Level 1 of the time window (Figure 4.5(a), (b) and
(c)). When Summary 3 is ready, Level 1 of the time window is full (Figure 4.5(c)). We will *merge* the summary objects on the filled level, expand the time window with one more level, and insert the merged summary objects into the expanded level. In Figure 4.5(d), Summary 1 and 2 are merged and moved to Level 2, and Summary 3 is inserted into Level 1. Similarly, Summary 4 is inserted into Level 1 (Figure 4.5(e)). When Summary 5 is ready, we first merge Summary 3 and 4 into a new summary object Summary 3-4 and insert it into Level 2 of the time window and place Summary 5 on Level 1 (Figure 4.5(f)). Note that the product of the merging operation (e.g. Summary 1-2 and Summary 3-4) must also fit in the constant memory budget $M$.

Using the pyramidal time window, the total memory footprint will grow logarithmically. More importantly, historical data is more compressed than recent data, so the summary is more accurate in recent time intervals. Next, we discuss the key operations of maintaining the time window: merging two summaries and maintaining time information.

**Merging Two Summary Objects** Given two summary objects $S_1$ and $S_2$, the corresponding patterns and the transactions represented by the patterns are $S_1 = (P_1, T_1)$ and $S_2 = (P_2, T_2)$. Both $S_1$ and $S_2$ are produced by the SPUR algorithm and can fit in a constant memory budget $M$. When merging $S_1$ and $S_2$ in the time window (Figure 4.5), we want to merge them into a new summary $S = (P, T)$ that can also fit in $M$. We first combine the pattern sets $P_1$ and $P_2$ to a new pattern set $P'$ by removing the duplicate patterns, representing all the transactions in $T_1$ and $T_2$ with a unified alphabet $P'$. Then we merge the transaction sets $T_1$ and $T_2$ into a new transaction set $T'$ to get a new summary $(P', T')$.  

87
Figure 4.5: Maintaining pyramidal time window

However, we cannot guarantee that \((P', T')\) can fit in the memory budget \(M\). More compression is needed to meet the budget. We rely on the output of SPUR to perform the compression. Remember the SPUR algorithm outputs a utility value for each pattern. This value measures the compression performance of a pattern. We use a greedy strategy to reduce the memory space of \((P', T')\). We sort the pattern utility values from low to high. We start with dropping the low utility patterns and removing them from the transactions they cover. This process will stop when the total size of the patterns and transactions left is below \(M\). We output the final merged summary \(S = (P, T)\) as our result. D-SPUR uses this merging algorithm to maintain the pyramidal time window in Figure 4.5. After merging two summaries, however,
the time information of the messages is lost since we cannot distinguish the messages between two batches. We will address this problem next.

**Maintaining Time** It is necessary to maintain time information for the transactions in a summary, otherwise the summary cannot effectively answer a query about an arbitrary time interval. For example, if the state for the summary is as shown in Figure 4.6, and we want to retrieve messages in batches 3 to 6, the query cannot be answered if no time information is stored with each transaction as the messages in batches 1 to 4 and 5 to 8 are all merged together.

![Figure 4.6: An example of the pyramidal time window](image)

When we compress individual batches to summaries with SPUR, the transactions in the summary are represented by the patterns selected by SPUR. It is likely that two similar but non-identical transactions will be represented by the same set of patterns and they will essentially be identical in the summary. Instead of storing them separately, we store distinct transactions in the summary and associate a count with each transaction to indicate how many times a transaction appeared in a batch.

When D-SPUR merges summaries of two adjacent batches, the merging operation combines their transaction sets. If two transactions contain the same set of patterns, they must be from two different batches, because within a batch, we only keep distinct transactions. Instead of summing the count of these two transactions, we could
concatenate their counts in time order and form a time series with two points. As D-SPUR combines more summaries, we concatenate more points to each transaction. A time series that spans batches (i.e. the red dashed line in Figure 4.7) is therefore formed for each transaction, enabling reconstruction of the exact count in any time interval.

However, this time series will grow linearly with time, which will violate the memory budget constraint. We therefore approximate (compress) the time series using a constant number of linear regression lines (blue line in Figure 4.7). Whenever the number of regression lines exceeds a constant $k$ after merging summary objects, D-SPUR will compress the two series for all transactions. We employ methods proposed by Palpanas et al. [84] for the purpose of compressing the time series. Their methods can find the optimal approximation when concatenating two adjacent time series.
4.3 Compression of Network Topology with G-SPUR

Next, we present G-SPUR, a variant of the SPUR algorithm to compress the topological data of networks. We assume that the network topology is dynamic, however, relatively stable compared to the streaming network content, because the frequency of users adding or deleting friends is much lower than posting new messages. Thus, we can take snapshots of the network at different times, e.g. one snapshot per week, and compress each snapshot separately. The complete topological information of a social network snapshot can be modeled as a directed graph $G = (V, E)$, where each node $v \in V$ represents a user and an edge $(v, u) \in E$ indicates user $u$ is a follower of $v$ in the social network. The storage space of such a graph in memory is proportional to the number of edges. Given the hundreds of millions of users and billions of edges in modern social networks, e.g. Twitter and Facebook, it requires large amount of memory spaces to store the topological structures of these networks. To serve queries related to user link structures in real-time, it is desirable to have a compact summary of the user connections in the main memory.

Our solution to this problem is to represent the adjacency list of a user as a transaction of items, where each item is a follower of this user. Then the entire social graph $G$ can be seen as a batch of user follower transactions. We would like to apply our SPUR algorithm to a batch of such transactions and compress its storage space.

However, the SPUR algorithm is not directly applicable to the graph summarization problem for the following reasons. First, the SPUR algorithm produces a lossy compression of the transactions because the infrequent items will be dropped in the summary, which corresponds to dropping edges in the network. However, dropping of edges in the graph data can make the graph disconnected and introduce errors
to graph mining algorithms that solely depend on the link structure of a graph, e.g. PageRank and Random Walk with Restart. Second, the SPUR algorithm needs to find frequent patterns from the dataset. The mining process is not scalable to a dataset with hundreds of millions of transactions and billions of items.

We propose the G-SPUR algorithm with two modifications of SPUR to enable lossless and fast summarization of large-scale graph data. Given a graph $G$ and a support threshold $\sigma$ used by SPUR, we first use a separate graph $G_{\text{infreq}}$ to preserve the infrequent edges dropped by the frequent pattern mining step of the SPUR algorithm. In the SPUR algorithm, the graph $G$ is represented by a database of transactions, where each transaction is the adjacency list of a vertex and items in the transaction are the neighbors connected to the vertex. With the support threshold $\sigma$, the SPUR algorithm will drop the neighbors whose numbers of adjacent edges are below the threshold. In G-SPUR, we use a separate graph $G_{\text{infreq}}$ to preserve these infrequent edges and leave the frequent edges in a graph $G_{\text{freq}}$ for further processing. If $G$, $G_{\text{infreq}}$ and $G_{\text{freq}}$ are represented as adjacency matrices, we can see the above process decomposes $G$ as the sum of $G_{\text{infreq}}$ and $G_{\text{freq}}$. Figure 4.8 shows this decomposition. Because $G_{\text{infreq}}$ only contains the edges connected to vertex with in-degree below the support threshold, we do not expect that it would consume large amount of storage space. Therefore, we can directly store it as a sparse matrix in memory.

After the above separation of infrequent and frequent edges, we can run SPUR algorithm on $G_{\text{freq}}$ without loss of information because all edges in $G_{\text{freq}}$ are frequent. The SPUR algorithm will compress $G_{\text{freq}}$ to a pattern set $P$ and a transaction set $T$. Each pattern $p \in P$ contains a set of vertices from $G_{\text{freq}}$. Each transaction $t \in T$ contains a set of patterns from $P$, corresponding to the compressed representation for
the original adjacency list. To reconstruct the adjacency list of a vertex from \( G_{freq} \), we can take the union of patterns in a transaction. In a binary sparse matrices representation of the pattern set \( P \) and transaction set \( T \), the SPUR algorithm essentially decomposes the frequent graph \( G_{freq} \) to the product of transaction set \( T \) and pattern set \( P \) (shown in Figure 4.9). Here the total storage of \( T \) and \( P \) will be much smaller than the original \( G_{freq} \).

By combining the infrequent graph \( G_{infreq} \) with the decomposition of \( G_{freq} = T \times P \), we can reconstruct the original graph \( G \) as \( G = G_{infreq} + T \times P \). Using \( G_{infreq} \), \( T \) and \( P \), we can store the original graph \( G \) with smaller storage size and reconstruct \( G \) without information loss.

Note that \( G_{freq} \) contains most of the edges in the original graph \( G \). Therefore, when graph \( G \) represents the user link graph of a large social network, the input to the SPUR algorithm will be as large as hundreds millions of transactions and billions of items. To maintain a scalable solution, we use minwise independent hashing [22] to partition the data into small samples. The SPUR algorithm will operate on each partition independently and produce a summary for each partition. We can generate the final solution by merging the pattern sets and transaction sets from the partitions.
This method has been used by Buehrer et al [25] to improve the scalability of large-scale web graph compression problems.

\[
\begin{array}{ccc}
G_{\text{freq}} & = & T \\ & \times & P
\end{array}
\]

Figure 4.9: Compress graph \(G_{\text{freq}}\) into transaction set \(T\) and pattern set \(P\).

4.4 Experimental Results

Next, we present results for an extensive set of experiments we conducted to evaluate our stream and graph summarization methods. We considered several other algorithms as candidates for our baseline. Methods that use probabilistic models to summarize patterns [112] are not efficient enough to handle the data influx rate we would like to handle. We therefore omit comparisons with these methods. We compare \(SPUR\) with the following algorithms: a) CDB [116] that uses rectangles to cover a transactional database b) RPMine [117] that tries to cluster the patterns and use the cluster centers to cover the remaining patterns. c) Krimp [103] that generates a static ranking of all patterns first and then summarizes the data using the top ranked patterns. Neither CDB nor RPMine can compress streaming data. Therefore, the \(D-SPUR\) algorithm is compared only with StreamKrimp [65], the streaming version of Krimp. For the G-SPUR algorithm, we present the compression performance of
several large web graphs and the follower–followee graph of Twitter. To show the benefits of graph compression in speeding up graph mining kernels, we also implement the PageRank algorithm using the summarized graph with CPUs and GPUs.

### 4.4.1 Dataset and Setup

We gathered 2100 hours of Twitter message streams from June to September in 2010\(^7\). The total data size is 5.9GB. For our evaluation, we partition the message stream into 1-hour batches. There are 100,000 messages per batch and 8 words per message on average after stop word removal and word stemming. We also crawled the follower lists of all the users in the above message stream\(^8\). We construct the follower–followee graph of Twitter from this dataset. There are about 131 million vertices and 3.8 billion directed edges.

All experiments were performed on a desktop machine with dual boot Red Hat Linux 6 and Windows 7 operating systems. The machine is equipped with an Intel i7 3.4GHz CPU and 16GB of main memory. Except for Krimp and StreamKrimp for which only Windows binaries are available, all algorithms were executed under Linux. All algorithms were implemented in C++.

### 4.4.2 Batch Compression with SPUR

In this section we present performance results for compressing a batch of messages to a summary object. Note that we only present results for batch summarization and not for the summary merging procedure. The SPUR algorithm is compared with the three baseline algorithms (CDB, RPMine and Krimp). The windows executable

---

\(^7\)As provided by Twitter, it is a 15% random sample of all messages.

\(^8\)Some of the users’ follower information is not available because of their privacy settings.
for Krimp and the source code for CDB and RPMine were obtained from the authors’ websites. As the baseline algorithms cannot compress to a target memory budget, we relax the memory budget constraint in our algorithm by summarizing a batch until the utility of the top pattern is negative. We present results for processing the first 100 batches since they are sufficiently representative of the entire data. Our comparative study focuses on three aspects: execution time, false positive rate (i.e. compression quality) and compression ratio.

**Comparison with CDB and RPMine**

We first compare SPUR with CDB and RPMine as they all support the tradeoff between false positive rate and compression ratio. Support level is set to 0.01% for all three methods. We set a low support level because we want the summary to cover as many topical words as possible. We set the maximum false positive rate for SPUR and CDB to 0.1. For RPMine, the pattern cluster tightness parameter is also set to 0.1.

![Figure 4.10: Running time: SPUR vs. CDB vs. RPMine](image-url)
Figure 4.10 presents the running times for the three methods in log scale. It is easy to see that our method is significantly more efficient – it is at least one order of magnitude faster than RPMine, and two orders of magnitude faster than CDB. SPUR is able to process an hours worth of data in less than one minute, lending itself to our requirement of being able to process the stream in real-time.

![Graph showing false positive rates for SPUR, CDB, and RPMine](image)

Figure 4.11: False positive: SPUR vs. CDB vs. RPMine

We next evaluate the algorithms’ false positive rate. Ideally, we want to introduce as few false positives as possible when reducing the data size. We therefore measure the actual false positive rate each method exhibits when given the same parameters for support threshold and maximum false positive rate. Figure 4.11 presents the false positive rate for each batch of the stream. Again, the performance of SPUR is solid – its false positive rate never exceeds 0.005. In contrast, CDB constantly suffers from a higher false positive rate, which is close to the maximum specified value. We also find that RPMine is susceptible to changes in data size, as suggested by the two spikes.
at batches 41 and 83. With relatively small data size for both batches, false positive rates reach 0.365 and 0.399 respectively.

![Compression Ratio: SPUR vs. CDB vs. RPMine](image)

Figure 4.12: Compression Ratio: SPUR vs. CDB vs. RPMine

As for the compression ratio defined as the ratio of original data size over the size of the summary (the higher the better), our method performs midway between CDB and RPMine with a range of 1.6 to 2 (see Figure 4.12). Again note the relatively low compression ratio for RPMine for batches 41 and 83. We conjecture that RPMine tends to cluster singleton patterns with longer patterns for these cases, leading to less reduction in size and a higher false positive rate. Note that the compression quality for our method is more robust against changes in data size, which is a favorable property.

All algorithms achieve exactly the same false negative rate as this only depends on the support level that is set to be same for all algorithms.
Comparison with Krimp

For this set of experiments, we compare Krimp with SPUR. We compare Krimp separately from CDB and RPMine for two reasons. First, it does not allow for false positives during compression. Second, its running time is very large when support level is as low as 0.01%. Therefore, we set the support level to 0.03% in the following set of experiments. For our method, we set the maximum false positive rate to 0.0.

![Running time: SPUR vs. Krimp](image)

Figure 4.13: Running time: SPUR vs. Krimp

The running times and compression ratios for the two algorithms are shown in 4.13 and 4.14. Our method is always at least 10 times faster than Krimp, and it also bests Krimp in terms of compression ratio.

4.4.3 Stream Summarization with D-SPUR

In this section, we present the performance of the D-SPUR algorithm on summarizing the Twitter message stream. There is no streaming version for CDB and
RPMine, thus we only compared D-SPUR with StreamKrimp at support level = 0.03% and false positive rate = 0. For D-SPUR, the memory budget $M$ is set to 1.2 MB as we find it yields low false negative error rate in the SPUR summarization step. We use 5 linear segments to approximate the time series of the transactions. This configuration leads to a compression ratio of approximately 50% for each batch.

We evaluate the algorithms in three aspects: execution time, memory footprint, and quality of the summary.

**Execution time:** Figure 4.15 shows the running times for D-SPUR and StreamKrimp. The x-axis represents the batch number and the y-axis shows the time different stream summarization algorithms take to summarize the input stream. We can see D-SPUR is much faster than StreamKrimp at the support level of 0.03%. Even at a lower support level of 0.01%, D-SPUR only takes approximately 40 minutes to summarize 100 hours worth of Twitter messages. Our method can thus process the data faster than it arrives, which is desirable for real-time stream processing.
Figure 4.15: Running time: stream summarization

Figure 4.16: Memory footprint: stream summarization
**Memory footprint:** As for memory consumption, we plot the storage size of the raw stream together with the memory footprint for different summarization algorithms in Figure 4.16. We can see that the memory footprint for D-SPUR grows very slowly – asymptotically it grows logarithmically in the size of the input. Furthermore, the size of D-SPUR’s summary is 8 times smaller than the raw data and 3 times smaller than the summary produced by StreamKrimp. This is because the merging of two summaries in D-SPUR is based on the pyramidal time window. If one does not use the pyramidal summarization scheme, the memory footprint grows linearly (see the green line in Figure 4.16). Also note that StreamKrimp has an even higher memory footprint than our SPUR algorithm. When a new batch of transactions streams in, the StreamKrimp algorithm first tries to compress the transactions with the codetable used for the previous batches. The old codetable is not always suitable for compressing the new batch and often it is forced to rebuild a new codetable for the new batch. In effect, it ends up maintaining separate summaries for individual batches.

**Quality:** We evaluate the summary quality by querying the summary with a randomly generated query workload. The query is in the format of a keyword \( w \) and a timestamp \( t \). \( w \) and \( t \) are sampled uniformly at random from all words above the support threshold and all time stamps. We do not consider infrequent words because they are dropped by the frequent pattern mining algorithm and can never be retrieved by any frequent pattern based summarization algorithms. All messages containing keyword \( w \) at time \( t \) are reconstructed and returned as the query result. We represent the result as a multiset of words, and compute the Jaccard similarity between the query result and the original messages that contain \( w \) at time \( t \) to measure query
accuracy. We expect the query accuracy to be high for both recent data and frequent keywords.

![Figure 4.17: Query accuracy (T=62)](image)

We present the distribution of query accuracy over time and word frequencies. Figure 4.17 presents the distribution of query accuracy over time and word frequencies where system time T is 62. The y-axis represents the query time stamps aligned with the pyramidal time window at T and x-axis represents the frequency of the query keywords. The color represents the accuracy at a specific time and word frequency, the darker the better. We can see on recent data, e.g. T = 61 – 62, the accuracy is high on all words; on historical data, e.g. T = 1 – 32, the accuracy is higher on frequent words than infrequent words; the overall accuracy on the recent data is higher than the historical data; and the accuracy on frequent words is generally higher than infrequent words.
Figure 4.18: Query accuracy (T=30)

Figure 4.18 shows the distribution of query accuracy when we rewind the system time $T$ to 30. We observe the same trend as the accuracy on recent data for frequent words is higher than that on historical data for infrequent words. Compared with the summary at system time 62 (Figure 4.17), we can see the overall query accuracy for batch 1 to 30 is higher, except for the infrequent words for batch 1 to 16. At system time 62, our pyramidal time window compresses the historical data (batch 1 to 30) more, so query accuracy on those batches is low. But at system time 30, the summary can keep more information and is more accurate.

4.4.4 Speeding up Graph Mining with G-SPUR on CPU and GPU

In this section we present experimental results of the G-SPUR algorithm. First, we show the compression performance of the G-SPUR algorithm on large-scale web and social graphs. We ran G-SPUR on the four web graphs in Table 3.2 and the follower–followee graph of Twitter we crawled. We use min-wise hashing to cluster the graph
into partitions with partition size less than 1000 and we ran SPUR algorithm on each partition with absolute support value at 5. Figure 4.19 shows the compression ratio of the G-SPUR algorithm on these graphs. We can see that the G-SPUR algorithm can compress the storage size of large-scale web graphs to as low as 4 times smaller than the original graphs. In the Twitter follower – followee graph, our G-SPUR algorithm can still reduce the storage size by half.

![Compression Ratio Chart](chart.png)

Figure 4.19: Compression ratio of G-SPUR on web and social network graphs.

Previous work [57] shows that graph mining algorithms such as PageRank and SALSA can be directly computed from compressed graphs and the performance can be improved because the total number of computations can be reduced due to the compression. Next, we will present how the compressed graphs produced by G-SPUR can speed up the PageRank algorithm on large graphs. From Equation 2.1, we can see that the PageRank algorithm can be implemented by iteratively calling the sparse matrix and vector multiplication (SPMV) kernel on a graph $G$. Our G-SPUR
algorithm essentially decomposes a graph $G$ to the form of $G = G_{\text{infreq}} + T \times P$. Figure 4.19 shows that the total storage size of $G_{\text{infreq}}, T$ and $P$ is much smaller than $G$. Because $G_{\text{infreq}}, T$ and $P$ can all be stored as sparse matrices, we can directly implement the PageRank algorithm as iterative SPMV on $G_{\text{infreq}} + T \times P$.

![Figure 4.20: CPU speed up of PageRank on compressed graphs.](image)

Next, we present some experimental results of speeding up the PageRank algorithm with G-SPUR. On CPU, we store the three sparse matrices $G_{\text{infreq}}, T$ and $P$ in CSR format and perform the matrix and vector multiplication. Figure 4.20 plots the speed-up numbers of compressed graph over uncompressed on the four large web graphs and the twitter social graph. We can achieve from 1.4x to 2.6x speed-ups on these dataset. On GPU, we store the three sparse matrices in our optimized composite storage format. To compare with the performance of our multi-GPU SPMV kernel, we distribute matrices $G_{\text{infreq}}$ and $T$ to multiple GPUs, and each GPU will keep a copy of $P$ because it is needed by all nodes of the GPU cluster. Figure 4.21
plots the speed-up numbers of the five datasets on GPUs. The PageRank algorithm can achieve from 1.1x to 2.2x speed-ups on the compressed graph over the original graphs. The major computational cost of the PageRank algorithm is the iterative SPMV kernel whose running time is proportional to the storage size of the graph. Our G-SPUR algorithm can effectively compress the size of the graph by decomposing it into three smaller matrices. Therefore, we can conclude that the speed-ups of the PageRank algorithm come from the compression of the graphs by the G-SPUR algorithm.

![GPU Speedup Graph](image)

Figure 4.21: GPU speed up of PageRank on compressed graphs.

4.5 Conclusions

We proposed an efficient summarization framework, which can incrementally build summaries of Twitter message streams as well as follower–followee graphs with one-pass over the data. We developed a novel algorithm to compress social network
content and topology with low compression ratio, high quality and fast running time. The memory footprint of our stream summarization algorithm grows approximately logarithmically with time. Our summary allows one to issue queries to retrieve messages over arbitrary time intervals. Our graph compression algorithm can dramatically reduce the storage size of network topology information and can speed-up a series of graph mining kernels.
Chapter 5: Complex Querying and Mining of Social Network Content and Topology

In Chapter 4, we introduced our novel pattern driven approach for summarization of the streaming content and network topology from modern social networks. The summary we built can be used to efficiently support complex mining queries for real-time analytics. In this chapter, we will present the query processing component of our large-scale network analytics framework.

With the summaries of the streaming content and social topology of social networks, there are two categories of mining queries that can provide insights of user behaviors in the social networks and can be efficiently answered by our proposed large-scale network analytics framework. First, with the summary of the dynamic network content, we can answer queries about the popular topics in the network content and how such topics evolve over time. Second, with the compressed network topology, we can also answer queries about how the popular topics spread through the users’ social connections.

In the following sections, we will formalize the above queries and then describe in detail how we can answer these queries using the compact storage of the social network content and topology we introduced in Chapter 4. We will also present some example queries and results from the Twitter social network.
5.1 Topic Event Detection Queries

In this section, we present an analytical tool TED-SPUR (Topic and Event detection with D-SPUR), to support complex queries on dynamic network content data. The input data is in the format of text message streams. We run our D-SPUR algorithm to build a summary of the message stream in memory. The summary is used to approximately reconstruct the original messages.

5.1.1 Live Analytics with TED-SPUR

One typical example of analytical query on text data is to discover topics [13, 91, 104]. Under the dynamic setting of message streams, it is also important to model the evolutionary behaviors [5] of topics. With the in-memory summary produced by D-SPUR, we can approximately reconstruct the original messages to perform those two tasks on arbitrary time interval(s).

We employ an implementation of non-negative matrix factorization (NMF) algorithm [104] to find the topics from the tweets within a query time interval, as NMF has been shown to handle sparse input well. The main idea is to factor a $d \times w$ document-word matrix into two non-negative matrices, the first of which a $d \times k$ document-topic matrix and the second a $k \times w$ topic-word matrix.

Another interesting task of analyzing tweet streams is to discover the evolution of the topics. For this task, we define a series of events to capture the appearance/disappearance of a topic, the growth/shrinkage/continuation of topic popularity and the merging/split/transformation of topic content. Given two time intervals $I_1$ and $I_2$, assume $I_2$ is after but not necessarily succeeding $I_1$. Let the mined topic sets from both intervals be $T_1$ and $T_2$ respectively. Each topic is further represented
as a word distribution vector $z$ and a scalar support value $sup$ indicating how many times it appears in the time interval. We use asymmetric KL-divergence [62] to capture the difference between two topics with regard to their content. The formalization of topic events is as follows:

**Appearance:** A topic $z$ appears in $T_2$ iff there is no topic $z'$ in $T_1$ such that $D_{KL}(z, z') < \gamma$, where parameter $\gamma$ measures the closeness of two topics. The intuition here is that if we cannot find a topic $z'$ in $T_1$ which is close enough to topic $z$, we will consider topic $z$ as a novel topic in $T_2$.

**Disappearance:** A topic $z$ disappears in $T_1$ iff there is no topic $z'$ in $T_2$ such that $D_{KL}(z, z') < \gamma$.

**Growth:** For two topics $(z, sup)$ in $T_2$ and $(z', sup')$ in $T_1$, if $D_{KL}(z, z') \leq \delta$, and $sup/sup' \geq 1 + \epsilon$ where $0 < \delta < \gamma$ and $\epsilon > 0$, then topic $z$ grows from topic $z'$. To explain, we find a topic $z$ in $T_2$ whose content is similar enough to topic $z'$ in $T_1$ while it is also more frequent than $z'$. Here the similarity threshold $\delta$ should be much lower than $\gamma$.

**Shrinkage:** For two topics $(z, sup)$ in $T_2$ and $(z', sup')$ in $T_1$, if $D_{KL}(z, z') \leq \delta$, and $sup/sup' \leq 1 - \epsilon$ where $0 < \delta < \gamma$ and $\epsilon > 0$, then topic $z'$ shrinks to topic $z$.

**Continuation:** For two topics $(z, sup)$ in $T_2$ and $(z', sup')$ in $T_1$, if $D_{KL}(z, z') \leq \delta$, and $1 - \epsilon < sup/sup' < 1 + \epsilon$ where $0 < \delta < \gamma$ and $\epsilon > 0$, then there is a continuation from $z'$ to $z$. To put it informally, if we can find two topics in $T_1$ and $T_2$ with similar enough contents and their strengths are also similar, then we regard the topic in $T_2$ as the continuation of the topic in $T_1$.

**Merging:** Given a topic $z$ in $T_2$, and a subset of topics $\{z'\}$ from $T_1$ such that each $z'$ satisfies $\delta < D_{KL}(z, z') < \gamma$. For any pair of topics $z'_1$ and $z'_2$ in $\{z'\}$, let the
weighted sum of distribution be $Z = \frac{z'_1 \times \sup'_1 + z'_2 \times \sup'_2}{\sup'_1 + \sup'_2}$. If $D_{KL}(z, Z) \leq \delta$, then we say $z$ is merged from $z'_1$ and $z'_2$. The merging event deals with topics which can neither be classified as growth/shrinkage/continuation from a previous topic, nor a newly appeared topic. The explanation is that if the content of a topic is similar enough to that of the weighted combination of two topics from previous time interval, then we consider the new topic to be merged from the previous two topics. Note that usually the resultant $z$’s from NMF are divergent from each other, making it unlikely that multiple pairs in \{z’\} merge into the same $z$.

**Split:** Given a topic $z$ in $T_1$, and a set of topics \{z’\} from $T_2$, such that $z'$ satisfies $\delta < D_{KL}(z', z) < \gamma$. For any pair of topics $z'_1$ and $z'_2$ in \{z’\} with the same definition of $Z$ as above, if $D_{KL}(Z, z) \leq \delta$, then $z$ is split into $z'_1$ and $z'_2$.

**Transformation:** If a topic $z$ in $T_2$ is not involved in any of the seven aforementioned events, $z$ is said to be transformed from topics \{z’\} in $T_1$, such that each $z'$ satisfies $\delta < D_{KL}(z, z') < \gamma$. The underlying intuition is that if a topic can find topics in the previous time interval which are partly similar to it and are not involved in any other event, then there is a content transformation to this topic.

By issuing multiple topic modeling queries over subsequent time windows and applying the above event detection algorithms, one can find evolutionary events over time.

5.1.2 Analytical Query Results with TED-SPUR

In this section, we present qualitative evaluations of our algorithm by presenting experimental results for answering topic modeling and event detection queries with our summary on the Twitter data. For these experiments, we apply D-SPUR on
<table>
<thead>
<tr>
<th>ID</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>south, africa, group, play, now</td>
</tr>
<tr>
<td>2</td>
<td>nowplay, yeah, people, twitter, follow</td>
</tr>
<tr>
<td>3</td>
<td>watch, big, play, fan, go</td>
</tr>
<tr>
<td>4</td>
<td>match, play, win, team, group</td>
</tr>
<tr>
<td>5</td>
<td>group, now, ENG, go, USA</td>
</tr>
<tr>
<td>6</td>
<td>live, home, real, watch, game</td>
</tr>
<tr>
<td>7</td>
<td>vuvuzela, watch, annoy, people, game</td>
</tr>
<tr>
<td>8</td>
<td>twitter, people, please, org, bit</td>
</tr>
<tr>
<td>9</td>
<td>vuvuzela, stadium, health, people, fifa</td>
</tr>
<tr>
<td>10</td>
<td>bit, photo, show, people, match</td>
</tr>
</tbody>
</table>

Table 5.1: Topics related to “world cup” from 5pm Jun 12th to 5pm Jun 13th.

subsets of the 2100-hour dataset to answer topic modeling and event detection queries. The data is divided into 6-hour batches with approximately 700K messages per batch. Support threshold is set to 0.01%, false positive rate to 0.1 and $M$ to 20 MB. For the topic modeling algorithm, we use $k = 50$ as the number of topics.

**Topic modeling:** We build a summary of data from June 11th to June 26th. All messages generated between 5pm, June 12th and 5pm, June 13th and match the query “world cup” are retrieved from the summary. This time interval is the second day after the World Cup in South Africa began. The top 10 topics returned by NMF are listed in Table 5.1, with each topic represented by the top 5 words of the topic’s word distribution. There are several topics detected from our summary that are very related to the keywords “world cup”. For example, “south africa” indicates the country where the World Cup was held; “vuvuzela” is a handy horn used by South African fans in the World Cup stadiums, and there was a popular debate about whether “vuvuzela” should be banned, because it made an annoying noise.
Note that the query time interval is also one day before the game between England and USA. The topic modeling algorithm detected a topic mentioning this game with the keywords “ENG” and “USA”. The detected topics show that our summary can approximately reconstruct the messages to support topic modeling queries.

**Event Detection:** Next, we give an example of an event detection query. A summary is built for data from June 26th to July 11th, and the “world cup” query is run on two time intervals: 5pm, July 3rd to 5pm, July 4th; 5pm, July 4th to 5pm July 5th. We first perform topic modeling on the retrieved messages. The resultant topics are listed in Table 5.2 and 5.3. Note that the first time interval is the day after the quarter-final game between Netherlands and Brazil, and the day before the quarter-final game of Argentina against Germany. From Table 5.2, we can see some related keywords of these two games, for example, *NED, BRA, ARG* and *GER* are abbreviations for the teams, and *Maradona* is the coach of Argentina, *Messi* and *Klose* are the players of Argentina and Germany respectively. In this time interval, we detected topics which discuss the NED vs. BRA game on the day before and predict the ARG vs. GER game. Similarly, the second time interval is the day after Germany beat ARG in the quarter-final and before the semifinal began. Table 5.3 lists the topics in the second time interval. We also detect topics which discuss previous day’s ARG vs. GER game, such as the keywords ARG, GER, Messi and Klose. There are also messages which predict that Germany can enter the final because they defeated Argentina, and messages that predict Spain (ESP) can enter the semi-final.

Next, we apply the event detection algorithms introduced in Section 5.1.1 on the topics of the two time intervals. We set the parameters in the algorithm to $\delta = 0.5$, $\gamma = 0.5$ and $\epsilon = 0.3$. The detected events are presented in Table 5.4.
Table 5.2: Topics related to “world cup” from 5pm Jul 3rd to 5pm Jul 4th.

<table>
<thead>
<tr>
<th>ID</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td><em>Messi, German, today, Klose</em>, goal</td>
</tr>
<tr>
<td>12</td>
<td>watch, game, show, tv, live</td>
</tr>
<tr>
<td>13</td>
<td><em>Maradona, start, say, people, match</em></td>
</tr>
<tr>
<td>14</td>
<td>fox, soccer, kickoff, down, <em>ARG</em></td>
</tr>
<tr>
<td>15</td>
<td>fan, tv, espn, score, twitter</td>
</tr>
<tr>
<td>16</td>
<td><em>ARG, GER, match, goal, Klose</em></td>
</tr>
<tr>
<td>17</td>
<td><em>ARG</em>, out, match, wow, go</td>
</tr>
<tr>
<td>18</td>
<td>thank, god, please, today, give</td>
</tr>
<tr>
<td>19</td>
<td>tweetphoto, party, nice, look, wow</td>
</tr>
<tr>
<td>20</td>
<td><em>NED, BRA</em>, match, play, fly</td>
</tr>
</tbody>
</table>

Table 5.3: Topics related to “world cup” from 5pm Jul 4th to 5pm Jul 5th.

<table>
<thead>
<tr>
<th>ID</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td><em>semifinal, tv, match, ESP</em>, go</td>
</tr>
<tr>
<td>22</td>
<td>watch, game, show, tv, live</td>
</tr>
<tr>
<td>23</td>
<td><em>german, final, Klose</em>, play, wow</td>
</tr>
<tr>
<td>24</td>
<td>www, home, look, win, play</td>
</tr>
<tr>
<td>25</td>
<td>tinyurl, home, play, game, fan</td>
</tr>
<tr>
<td>26</td>
<td><em>ARG, GER, Messi, goal, Klose</em></td>
</tr>
<tr>
<td>27</td>
<td><em>ARG</em>, die, game, team, play</td>
</tr>
<tr>
<td>28</td>
<td>auction, dasar, demi, goal, otw</td>
</tr>
<tr>
<td>29</td>
<td>www, live, show, please, home</td>
</tr>
<tr>
<td>30</td>
<td>haha, yeah, watch, gua, cool</td>
</tr>
<tr>
<td>Event Type</td>
<td>Topic ID</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Appear</td>
<td>21, 23, 24, 25, 28, 29, 30</td>
</tr>
<tr>
<td>Disappear</td>
<td>13, 14, 15, 18, 19, 20</td>
</tr>
<tr>
<td>Continue</td>
<td>12 $\rightarrow$ 22</td>
</tr>
<tr>
<td>Merge</td>
<td>(11, 16) $\rightarrow$ 26</td>
</tr>
<tr>
<td>Transform</td>
<td>17 $\rightarrow$ 27</td>
</tr>
</tbody>
</table>

Table 5.4: Event detection from the topics in Table 5.2 and 5.3

in the table are topic ids in Table 5.2 and 5.3. Among the events, the newly emerged topics about predicting the semifinal and final games are detected as appear; the topic about the NED vs. BRA game in the day before is categorized as disappear; two same topics about live TV coverage from the two time intervals are detected as continue event; the two topics about the ARG vs. GER game in the first time interval are merged to a single topic in the second time interval after Germany defeated Argentina; finally, a thread of topic about Argentina is transformed to a similar topic in the second time interval. We can see that our event detection algorithm can detect evolutionary events which are aligned with events in the real world. These events can help users understand the dynamics of the topics in Twitter messages.

5.2 Content and Time Aware Network Topology Queries

In the previous section, we introduced how we can find popular topics and the evolution of such topics from the compact summarization of the social network content. Besides the topics, another important analytical task is to investigate the network topology of the users who have written or read messages about a topic. Given a topic or keyword, example queries can be as simple as finding users who wrote or read this topic. People can gain more insightful information of the network topology if
we can find the social connections among these users. Furthermore, with these user connections, we can find which users are influential writers about this topic, whether there are any community structures among the users, and how such communication structures change dynamically over time.

We can use the compact storage of the network content and topology proposed in Chapter 4 to answer the above queries. The workflow of answering those queries can be divided into two major steps. In the first step, given a query keyword and time interval, we extract a subgraph of the entire network topology which contains all the users who either wrote or read a message about this keyword during the query time interval. This subgraph includes all users who express interests to the topic represented by the query keyword. In the second step, we run various graph mining algorithms on the extracted subgraphs to find influential users, community structures and dynamic patterns in the network topology. Next, we will show the methods and results of these steps.

5.2.1 Content and Time Aware Subgraph Construction

Next, we present our method of constructing a user subgraph given a query keyword and time interval. We achieve this task by querying on the summaries built by the D-SPUR and G-SPUR algorithms introduced in Chapter 4.

Given a content keyword, we first find the list of users who have written messages about this keyword during the query time interval. We can slightly modify our D-SPUR algorithm to fulfill this query requirement. The D-SPUR algorithm produces a dynamic summary of the network content. A sequence of pyramidal time windows are used to organize the summary objects of the message streams. Each summary
Figure 5.1: Modification to the D-SPUR algorithm

object contains a pattern set and a transaction set. The steps to retrieve the list of users who have written messages about a keyword during a time interval is as follow: First, we query the summary objects within the query time interval from the pyramidal time window; Second, in each summary object, we retrieve the patterns that contain the query keyword, and the transactions that include these patterns. These transactions represent all the messages containing the query keyword in the query time interval. Third, we need to find the writers of these messages. These users are who have written messages about the query keyword during the query time interval. Here we need to slightly change the D-SPUR algorithm to retrieve these users. In the D-SPUR algorithm, we store time series at each transaction to keep track of the count of a transaction at a given time point (Figure 4.7), but the users’ id information of each transaction is dropped. To preserve the users’ information, we modify the D-SPUR algorithm by adding the list of users’ ids at each point in the time series. Figure 5.1 illustrates this modification. Therefore, at a given time and a
specific transaction, we can know who the users that wrote the transaction are. With
the above steps, we can extract the complete list of the writers of messages with a
query keyword during a query time interval.

Note that we can modify the D-SPUR algorithm to support the above user related
querying. Our modifications will increase the memory size of the summary object,
because we add a list of user ids for each transaction at each time point. This will
make the memory footprint grow linearly with time instead of logarithmically as in
the original D-SPUR algorithm. Since we are trying to answer more complex queries,
we need to pay more storage cost to achieve our goals.

With the list of users $U$ who have written messages about a query keyword during
the query time interval, we then find their social connections by extracting a subgraph
of them and their followers from the network topology. We will use the compressed
network topology built by G-SPUR to find these social connections.

\[ G = G_{\text{infreq}} + T \times P \]

Figure 5.2: G-SPUR compression of Network Topology

Suppose a graph $G$ represents the original network topology which contains the
social connections among all users. Figure 5.2 shows that our G-SPUR algorithm
decomposes the adjacency matrix of $G$ into the adjacency matrix of a graph $G_{\text{infreq}}$
and the product of a transaction – pattern matrix $T$ and a pattern – item matrix
Here $G_{\text{infreq}}$ contains the infreq edges connected to vertices with in-degree below a support threshold. Given a list of users $U$, we need to extract a subgraph which contains the connections among the users in $U$ and their followers from $G$. In the original graph $G$, this subgraph corresponds to a subset of the rows $G(U,:)$ in the adjacency matrix of $G$, where each row represents the followers information of a user in $U$. For example, the three highlighted rows in the matrix $G$ in Figure 5.3 contains the social connections of three users and their followers. Because the G-SPUR decomposition can represent the adjacency matrix $G$ as $G = G_{\text{infreq}} + T \times P$, we can extract a subset of the rows $G(U,:)$ as $G(U,:) = G_{\text{infreq}}(U,:) + T(U,:) \times P$. This means we can get the rows of users in $U$ from matrices $G_{\text{infreq}}$ and $T$ first, then use the rows from $T$ to multiply with the pattern matrix $P$ and then add to the rows from $G_{\text{infreq}}$. For instance, to get the three highlighted rows from $G$ in Figure 5.3, we first get three rows from $G_{\text{infreq}}$ and $T$ respectively, then multiply the three rows from $T$ with $P$, and then add the result to the three rows from $G_{\text{infreq}}$, the final results will be equivalent to the three highlighted rows in the original $G$. Therefore, we can efficiently extract a subgraph of the network topology from the D-SPUR and G-SPUR summaries by querying a content keyword and a time interval.

Figure 5.3: Extract subgraph with G-SPUR
In this section, we present how to extract a subgraph from network topology based on a query keyword and time interval from the summaries built by a modified version of our D-SPUR algorithm and our G-SPUR algorithm. With a keyword \( w \) and a time interval \( t \), we slightly change the summary produced by the D-SPUR algorithm to return a list of users \( U \) which contains all the users who wrote messages about the keyword \( w \) during time interval \( t \). Given the social network topology information \( G \), we use \( U \) to extract a subgraph of \( G \) which contains all the connections among the users in \( U \) and their followers. We can use \( G(w,t) \) to represent this subgraph of the graph \( G \). In the next section, we will introduce how to apply various graph mining algorithms on such content and time aware subgraphs. We will also show some experimental results of these mining algorithms on the Twitter social network dataset.

### 5.2.2 Mining Algorithms on Subgraphs of Network Topology

With a subgraph of a network topology conditioned on a query keyword in a query time interval, we can perform static analysis such as finding relevant users or communities of users on a topic during the query time interval. Given the subgraphs of multiple time intervals, we can also study the dynamic evolution of user communities. In this section, we will introduce some example mining queries to perform these tasks under our proposed framework. We use the PageRank algorithm to rank the users in a subgraph and measure their relevance to the query topic. To find user communities from a static subgraph, we use the MLR-MCL algorithm to locate dense user communities. The event detection algorithm is used to find the evolution of such
communities. We introduce the details of our methods and results of performing the above tasks in the following sections.

**PageRank**

To rank the users’ importance or relevance to a topic keyword $w$ during a time interval $t$, we first extract a subgraph $G(w, t)$ of the entire network topology $G$. The directed subgraph $G(w, t)$ captures the social connections among all the users who wrote messages of the topic keyword $w$ in time interval $t$. We use the PageRank algorithm to rank the relative importance of the users in subgraph $G(w, t)$. We iteratively run Equation 2.1 on the adjacency matrix of $G(w, t)$. The computation of the PageRank algorithm can be accelerated by the GPU based high performance computing platform introduced in Chapter 3.

Next, we will show some experimental results of such PageRank queries on the Twitter social network data we crawled from June 2010 to September 2010. The detail of this dataset is introduced in Chapter 4. We extracted the subgraph of the entire network topology of users who mentioned the Twitter hashtag “#travel” in the summer of 2010. We run the PageRank algorithm on this subgraph and rank the user accounts by their PageRank value from high to low. Table 5.5 lists the top 20 ranked Twitter accounts about the keyword “travel” in the summer of 2010 and their account profile messages which generally describe each account.

The top ranked twitter account about travel is “Lonely Planet” which is the largest travel guide book and digital media publisher in the world. Their travel guidebooks are widely read by travelers for advice, tips and destination information. Our findings also show that their tweets are very popular in online social media. By
<table>
<thead>
<tr>
<th>Rank</th>
<th>Screen Name</th>
<th>Account Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lonely Planet</td>
<td>Tweeting (&amp; retweeting) the best in travel</td>
</tr>
<tr>
<td>2</td>
<td>GWPStudio</td>
<td>Photography, Socialmedia &amp; sharing ... Love to travel &amp; connect with people</td>
</tr>
<tr>
<td>3</td>
<td>American Airlines</td>
<td>The official channel of American Airlines</td>
</tr>
<tr>
<td>4</td>
<td>Sean Gardner</td>
<td>Digital Media Consultant</td>
</tr>
<tr>
<td>5</td>
<td>Gary Arndt</td>
<td>Traveler, blogger and photographer. A one man National Geographic. Been to over 100 countries and all 7 continents</td>
</tr>
<tr>
<td>6</td>
<td>Travelzoo</td>
<td>Travelzoo deal experts research and evaluate thousands of deals each week, selecting only the best to endorse and publish</td>
</tr>
<tr>
<td>7</td>
<td>SmarterTravel</td>
<td>SmarterTravel.com is the largest online travel resource for unbiased travel news, deals, and timely expert advice</td>
</tr>
<tr>
<td>8</td>
<td>WhereIveBeen</td>
<td>Travel industry’s leading social networking travel platform</td>
</tr>
<tr>
<td>9</td>
<td>TravelDeals</td>
<td>Use Twitter to save on travel in popular locations. Get a customized feed of travel deals near you</td>
</tr>
<tr>
<td>10</td>
<td>USA Today Travel</td>
<td>USA TODAY Travel offers the latest travel news, deals and consumer features about flights, hotels, cruises and destinations</td>
</tr>
<tr>
<td>11</td>
<td>Andreas Susana</td>
<td>A guy from Austria, who writes about his trips and his website concerning books, castles, museums, exhibitions, historic places and trains.</td>
</tr>
<tr>
<td>12</td>
<td>Melvin</td>
<td>Love to travel, to discover the world, to travel free &amp; untroubled &amp; still be informed like an insider! For Travelers, By Travelers! &amp; founder of Traveldudes. Citizen of the world</td>
</tr>
<tr>
<td>13</td>
<td>JD Andrews</td>
<td>World Traveler, Dad, 3xEmmy winner, Video, Adventure, Photographer, love dogs, Sharing &amp; Caring, Travel &amp; Social Media</td>
</tr>
<tr>
<td>14</td>
<td>British Airways</td>
<td>Official British Airways global account</td>
</tr>
<tr>
<td>15</td>
<td>Get a Travel Deal</td>
<td>I find the best travel deals so you don’t have to. Life’s Short Travel Often!</td>
</tr>
<tr>
<td>16</td>
<td>Eagles Nest Wine</td>
<td>San Diego’s Medal winning-ist Boutique Winery! Share an Authentic Wine Lifestyle with us! Luxury Lodging &amp; Wine Tourism, Sustainable &amp; Organic Farmers, Veterans</td>
</tr>
<tr>
<td>17</td>
<td>Chicago Events</td>
<td>Real-time local buzz for live music, parties, shows and more local events happening right now in Chicago!</td>
</tr>
<tr>
<td>19</td>
<td>TravelGreen</td>
<td>Tips for sustainable travel and green living. Exploring the world, trying new foods &amp; being green.</td>
</tr>
<tr>
<td>20</td>
<td>Tourism Malaysia</td>
<td>The official Tourism Malaysia Twitter account.</td>
</tr>
</tbody>
</table>

Table 5.5: Top 20 ranked users about the keyword “travel”
scanning through the top 20 list in Table 5.5, we can classify these popular Twitter accounts in travel into the following categories.

- Free information sources that people follow to find and share travel information, such as #1 Lonely Planet, #8 WhereIveBeen, #10 USA Today Travel and #19 TravelGreen.

- Travel deal websites, including #6 Travelzoo, #7 SmarterTravel, #9 TravelDeals and #15 Get a Travel Deal. These results are from a subgraph queried during the time of summer 2010. We know that people often have their vacation trips in summer and they want to reduce their travel expenses. Therefore, it is expected that those travel deal websites are active and popular on Twitter during the summer.

- Airline companies such as #3 American Airlines and #14 British Airways, because information of air transportation is a huge factor for travelers to plan their itinerary.

- Interesting travel destinations including #16 Eagles Nest Winery, #17 Chicago Events and #20 Tourism Malaysia to promote their travel packages. Since our dataset is only a random sample from all tweets, we did not find any tweets written by accounts representing famous places of interest in our dataset.

- Famous individual bloggers to share their experiences. This category includes #2 GWPS Studio, #4 Sean Gardner, #5 Gary Arndt, #11 Andreas Susana, #12 Melvin and #13 JD Andrews. Some commonalities among these accounts are that they have large number of followers, they almost always follow back to
their followers, they also write a lot of tweets and post photos to share their own travelling experiences.

From the above example, we can see that by running the PageRank algorithm on the subgraphs of a social network, we can find popular and influential account representing organizations, companies or individuals related to a content keyword in a time interval.

**Clustering**

Given a subgraph $G(w,t)$ of the entire network topology $G$, we would like to find communities in such subgraphs to capture the topological relationships among the users who write or read the content keyword $w$ in time interval $t$. Instead of finding user communities in the global and static network topology, we project the topological information to a subspace of network content and time. This is much more meaningful for targeted advertisement or viral marketing through a social network, because users will have different interests in different topics and will form different community structures. In this section, we introduce a new type of complex query to find such communities. Given a content keyword $w$ and a time interval $t$, the query will return a clustering result $C$ of the active users who write or read messages about the keyword $w$ during time interval $t$ in the social network.

The general idea of answering such query is to first extract the subgraph $G(w,t)$, then apply graph clustering algorithms, e.g. MLR-MCL on the subgraph to find user communities. However, there are several challenges to execute the above process in a large-scale social network for the following reasons:
• **Scalability**: Depending on the popularity of the query keyword and the duration of the query time interval, the scale of the subgraph $G(w, t)$ can be as large as millions of vertices and edges. For example, a popular keyword and a long query time interval can include large number of users and their followers in a subgraph. Also when a hub node is included in a subgraph, the large number of followers of the hub node can dramatically increase the size of the subgraph. In this situation, it is difficult to run graph clustering algorithms and answer large number of queries efficiently.

• **Noise**: Previous work [113] has shown that there are two types of users: influential and auxiliary users in modern social network such as Twitter. The influential users are less than 1% of the Twitter population but produce more than 50% of the content, e.g. celebrities, writers and politicians. The auxiliary users are mostly fans, followers and readers of the influential users. The influential users usually have a large number of followers who are auxiliary users. The appearance of auxiliary users is challenging in community detection tasks, especially in a content aware graph such as our $G(w, t)$. Topologically the auxiliary users may form clusters probably because they know each other in real life. But this does not mean they share common interests of network content. It is more important to find the communities of influential users which are community kernels [113] and are more related to the network content. In some sense, the auxiliary users become the noise for finding community kernels of influential users. The large number of auxiliary users will also slow down the graph clustering algorithms.
• **Connectivity**: The influential users usually have a lot of followers but they rarely follow back. The connections among the influential users are weak which makes it hard to find dense communities if we do not include the auxiliary users. The auxiliary users who follow different influential users are essential to improve the connectivity of our subgraph $G(w, t)$.

• **Directionality**: The follower - followee relationships among Twitter users are directed. Therefore, our subgraph $G(w, t)$ is a directed graph. Satuluri et al [100] show that it is non-trivial to cluster directed graphs by using graph clustering algorithms designed for undirected graphs.

To overcome the above challenges, we use a simple but effective preprocessing approach to symmetrize the directed graph $G(w, t)$ and adjust the edge weight to improve connectivity among influential users and reduce the noise from auxiliary users. Then we call scalable graph clustering algorithm MLR-MCL on the transformed graph to find dense community kernels. The preprocessing steps are as follow:

1. **Symmetrization**: we use the bibliometric symmetrization [100] method to transform our asymmetric subgraph $G(w, t)$ to a symmetric graph $SG(w, t)$. Given the adjacency matrix $A(w, t)$ of $G(w, t)$, the bibliometric symmetrization essentially calculates the adjacency matrix of $SG(w, t)$ as $A(w, t)A(w, t)^T + A(w, t)^TA(w, t)$. This transformation will not only remove the directionality of edges in $G(w, t)$ but also add edges to vertices sharing similar set of in- or out- links. In $SG(w, t)$, edges can potentially be added between two influential users who are not directly connected, but share common followers. Thus, the connectivity among influential users is improved.
2. **Edge Re-weight**: The edges in $SG(w, t)$ are already weighted by the bibliometric symmetrization. But the weights are purely based on the topological information. We would like to incorporate the network content information to down-weigh the connections among auxiliary users who rarely contribute content to the network. The purpose of adjusting the edge weight is to remove the noise introduced by the large number of auxiliary nodes in the social network.

Given the keyword $w$ and a node $i$, we can have a weight $W(i)$ for node $i$ by counting the number of times user $i$ mentioned the keyword $w$ in the time interval $t$. Suppose we have an edge from node $i$ to node $j$, and the edge weight in the symmetrized graph $SG(w, t)$ is $SG_{w,t}[i][j]$. We adjust the weight of edge $<i, j>$ by multiplying $SG_{w,t}[i][j]$ with the sum of $W(i)$ and $W(j)$. In this way, we can construct a new weighted symmetric graph $WSG(w, t)$ with edge weight $WSG_{w,t}[i][j] = SG_{w,t}[i][j] \times (W(i) + W(j))$. The intuition of this edge re-weight method is that the node weight $W(i)$ captures how much interest user $i$ has on the keyword $w$. Therefore, we want to boost up the weight of edges connected to nodes with strong interest on the query keyword $w$, especially for the edges that both the follower and the followee express strong interests.

After the above preprocessing on our subgraph $G(w, t)$, we have a weighted undirected graph $WSG(w, t)$ where the nodes of similar influential users are connected together and the links to noisy auxiliary nodes are down-weighted. We can run scalable graph clustering algorithms such as MLR-MCL [99] to efficiently find dense communities of influential users.

Next, we will present some experimental results about graph clustering queries. Table 5.6 shows the representative users from two clusters we got when we query the
Table 5.6: Clustering results for the keyword “coupons”

<table>
<thead>
<tr>
<th>Cluster 1</th>
<th>Cluster 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrizeDrawsUK</td>
<td>coupons2grab</td>
</tr>
<tr>
<td>Deals4UK</td>
<td>slickwallet</td>
</tr>
<tr>
<td>TopUKDeals</td>
<td>CouponSpy</td>
</tr>
<tr>
<td>PiggyCodeUK</td>
<td>DirectCoupons</td>
</tr>
<tr>
<td>CodesUK</td>
<td>redtagdeals</td>
</tr>
<tr>
<td></td>
<td>CouponsInfo</td>
</tr>
<tr>
<td></td>
<td>CouponNet</td>
</tr>
<tr>
<td></td>
<td>Deals_Vista</td>
</tr>
<tr>
<td></td>
<td>CouponCodeFeed</td>
</tr>
<tr>
<td></td>
<td>internetshopper</td>
</tr>
<tr>
<td></td>
<td>SavvyPCDeals</td>
</tr>
<tr>
<td></td>
<td>Spaffin_5bay</td>
</tr>
</tbody>
</table>

subgraph for the keyword “coupons” in the time interval from July 1st, 2010 to July 31st 2010 in our Twitter dataset. We can see from the screen names of these users that they are all related to coupons and deals in online shopping. Quite a few online shopping and deal websites have such Twitter accounts for marketing purposes. The two columns in Table 5.6 list the influential users from the two clusters we found from the subgraph we extracted. There are a lot more auxiliary users who are followers of the users listed in the table. We omitted them here. We can see that the user names listed in cluster 1 are the Twitter accounts for online shopping websites in UK whereas the user names in cluster 2 are mostly in US with some global online shopping websites. Both cluster 1 and cluster 2 can be considered as community kernels because the user accounts have a lot of followers who are their customers. Also this cluster arrangement is reasonable because the accounts in different clusters have different follower populations. The followers of cluster 1 are mostly customers.
from UK whereas the followers of cluster 2 are mostly in US. Such clustering analysis is useful for online marketing with a targeted customer population.

**Event Detection**

The topic event detection query introduced in Section 5.1.1 shows key insights about how popular topics in the network content evolve over time. Given an evolving topic, it is also important to show how the social connections among the users who write or read about this topic evolve dynamically with the additional information of network topology. The ability to query such dynamic behavioral patterns can help people understand how popular topics spread through the social connections of the users.

In the previous section, we introduce our method of finding user communities in the subgraphs of network topology. Given such subgraphs from two time intervals, we can perform this clustering approach on the two subgraphs separately. With the event detection algorithm introduced in Section 2.1.3, we can study the evolution of such community structures over time. Next, we will introduce the event detection query to detect the dynamic behavior of the community structure evolution in the subgraphs. The format of the event detection query can be formally defined as: Given a query keyword $w$, and two query time intervals $t_1$ and $t_2$, find the community events between the subgraphs containing the users who write or read messages about the keyword $w$ in time interval $t_1$ and $t_2$.

To answer the above event detection query, we first extract the subgraphs that contain the users who write or read the query keyword $w$ in time interval $t_1$ and $t_2$. We denote these two subgraphs as $G(w, t_1)$ and $G(w, t_2)$. With the summary of the network content built by our D-SPUR algorithm and the compressed network topology
Top 20 user screen names

<table>
<thead>
<tr>
<th>Screen Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTV</td>
</tr>
<tr>
<td>JustinBieberFan</td>
</tr>
<tr>
<td>JBSource</td>
</tr>
<tr>
<td>JBieberBoy94</td>
</tr>
<tr>
<td>Bieber_projects</td>
</tr>
<tr>
<td>JDBWorldWide</td>
</tr>
<tr>
<td>mtvnews</td>
</tr>
<tr>
<td>TeamJBieberCrew</td>
</tr>
<tr>
<td>belieberbabes</td>
</tr>
<tr>
<td>RespectJustinB</td>
</tr>
<tr>
<td>TheBieberTweet</td>
</tr>
<tr>
<td>iBelieberrGuy</td>
</tr>
<tr>
<td>ThatBieberTeam</td>
</tr>
<tr>
<td>BieberWorldJDB</td>
</tr>
<tr>
<td>xSupport_JBMiley</td>
</tr>
<tr>
<td>JustinCrew</td>
</tr>
<tr>
<td>Bieber-swag</td>
</tr>
<tr>
<td>BelieberBoyX</td>
</tr>
<tr>
<td>AwwBieber</td>
</tr>
<tr>
<td>BieberTeamNY</td>
</tr>
</tbody>
</table>

Table 5.7: Continue event for the keyword “justinbieber”

contructed by our G-SPUR algorithm, we can use the approach introduced in Section 5.2.1 to extract these subgraphs. Then we can run the graph clustering method introduced in Section 5.2.2 to find user clusters from the subgraphs of each time interval respectively. Suppose \( C^1_i \) represents the cluster arrangement of subgraph \( G(w,t_1) \) and \( C^2_j \) represents the cluster arrangement of subgraph \( G(w,t_2) \), we can run the event detection algorithm introduced in Section 2.1.3 to detect evolutionary events such as continue, merge, split, form and dissolve events of communities.

In our dataset, the network topology is a static snapshot of the Twitter follower–followee graph. Due to this limitation, it is hard to detect form and dissolve events because these two types of events require that the sets of active users in \( t_1 \) and \( t_2 \) are exactly the same, then the subgraphs \( G(w,t_1) \) and \( G(w,t_2) \) will be very similar. There will only be slight difference in the edge weights. This will lead to similar clustering arrangements in \( t_1 \) and \( t_2 \), and there will be no form nor dissolve events by comparing these clustering arrangements. Next, we will show some example query results of continue, merge and split events from our Twitter social network dataset.

- **Continue Event**: We found an example of continue event about the search keyword “justinbieber” between the time interval from August 25th to August
27th and the time interval from August 28th to August 30th. The search keyword “justinbieber” is the screen name of the pop music star Justin Bieber in Twitter. He has a large number of die-hard fans as well as frequent exposure in the entertainment media. His screen name is very popular in the tweets of his fans and the entertainment news or gossips. In this continue event, the users in the cluster are active in both time intervals and put in the same clustering arrangement by the graph clustering process. Here the notion of “active” means a user either writes or reads a tweet about the query keyword. The screen names of some representative users from the user cluster in this continue event are listed in Table 5.7. There are totally 973 users in this large community of users. Here we only show the top 20 users with the most followers in Table 5.7. From the screen names, we can tell that most of the users are fans or fan groups of Justin Bieber, e.g. JustinBieberFan, Team JBieberCrew etc. We also see entertainment media such as MTV and mtvnews that give Justin Bieber continuous exposure to the public. From this example, we can see continue event can help us find the users who continuously give attention to a popular topic.

- **Merge Event**: An example of merge event about the hashtag “#madison-squaregaga” is detected between the time interval from June 27th to July 3rd and the time interval from July 4th to July 10th. Table 5.8 shows some of the representative user screen names involved in the clusters of this merge event. The hashtag “#madisonsquaregaga” stands for the pop music star Lady Gaga’s concert that took place on July 6th in Madison Square Garden, New York. Our
<table>
<thead>
<tr>
<th>June 27th to July 3rd</th>
<th>July 4th to July 10th</th>
</tr>
</thead>
</table>

Table 5.8: Merge event for the keyword “#madisonsquaregaga”
first query time interval is before the time of the concert and there are two separate clusters of users who wrote or read tweets about the concert. Most of the users in the clusters are fans of Lady Gaga. The concert happened within our second query time interval. In the second time interval, there are more users who actively wrote and retweet about this event. Especially Lady Gaga herself wrote a tweet with our query hashtag “#madisonsquaregaga” and was widely retweeted by her fans. Because of the boost in the volume of this topic, the subgraph becomes denser and more connected in the second query time interval. The two clusters in the first query time interval are merged by the newly activated common followees or followers of the two clusters. The dynamic change in the network content leads to the merge event happened in the network topology. To validate the above analysis, we plot the number of active users and the number of tweets about the query hashtag in the two query time intervals in Figure 5.4. We can see that the volume of the query keyword apparently increased in the second time interval in terms of the number of active users and the number of tweets. We can see that this merge event can topologically present the emerging of a popular topic such as Lady Gaga’s concert in Madison Square Garden.

• **Split Event**: A split event is detected about the hashtag “#oilspill” between the time interval from July 12th to July 18th and the time interval from July 19th to July 25th. The hashtag “#oilspill” refers to the oil spill accident happened in the Gulf of Mexico. The spill was caused by an explosion from a sea-floor oil gusher operated by BP and was considered the largest accidental marine oil spill in the history of the petroleum industry. This accident was
widely covered in all major news media. On July 15th, the gushing wellhead was finally capped after flowed unabated for about three months. This event happened in our first query time interval. We found a cluster of users that wrote tweets containing this hashtag in this time interval. The left column in Table 5.9 lists some representative user screen names in this cluster. These users can be divided into several categories:

(a) Parties involved in this accident. BP_America is the official Twitter account of BP, the company that causes this accident. Accounts like wildlife_fl, Visit Mississippi, Visit Florida, NewOrleans.Com and Louisiana Travel represent the places that are polluted by this accident. They can be seen as the victims in this incident.

(b) Major news media, such as cnnbrk, HuffingtonPost, washingtonpost, CNBC and so on.
Government agencies including whitehouse and CDCEmergency.

Environmental groups such as GreenPeace, NRDC, Oceana and etc.

In the second query time interval, this cluster splits into two smaller clusters. One of the clusters contains the parties involved in this accident, e.g. BP_America, Visit Mississippi and Visit Florida, and some news agencies such as HuffingtonPost. The other cluster only contains the environmental groups. This split event happens because of two reasons. First, the two government agencies White House and CDCEmergency were not active in the second query time interval. Both of them only write one tweet about the query hashtag in the first time interval and no further post in the second time interval. Second, some news agencies disappeared in the second time interval because they do not have follow-up coverage about this accident after the oil gusher was capped. Because these users are inactive, the subgraph in the second time interval becomes less connected and breaks into two clusters. From this split event example, we can see that split events are associated with the weakening of certain topics.

5.3 Conclusions

In this chapter, we proposed complex queries for large-scale network analytics. Without the need to access the original data, these queries can be efficiently answered by the compact summary of the network content and topology we built in Chapter 4.

First, we support queries about the popular topics in the network content during a time period. We process such queries by approximately reconstructing the network content in the query time interval from the summary built by our D-SPUR algorithm, then topic modeling algorithms can be used to find the topics from the reconstructed
content. Second, we define topic event to capture the dynamic evolution of such topics over time. Third, we propose content and time aware network topology queries to study topological information of a large-scale network. Conditioned on specific network content and a query time interval, we project the entire network topology to a subgraph. We study the relevance of network nodes to the query content in the time period by ranking the nodes with PageRank. We find community of network nodes by clustering the subgraph. We run event detection algorithms to detect the dynamic evolution of the communities. Finally, we present examples of the above query types on a real dataset from the Twitter social network.

To conclude, the queries we proposed play an important role in retrieving useful insights from large-scale network content and topology. The query results can not only provide meaningful knowledge about the data but also improve interactions with users.
Chapter 6: Visual Analytics

In this chapter we describe a visual-analytic tools to visually present the complex query results introduced in Chapter 5. To visualize the results of network content queries, our toolkit presents the detected topics, their trends over time, and the topic event to capture their evolutions. To visualize network topology queries, we develop a visual-analytic tool [119] for the interrogation of large-scale evolving interaction network data such as those found in social, bibliometric, WWW and biological applications. The tool we have developed incorporates common visualization paradigms such as zooming, coarsening and filtering while naturally integrating information extracted by a previously described event-driven framework for characterizing the evolution of such networks. The visual front-end provides features that are specifically useful in the analysis of interaction networks, capturing the nature of both individual entities as well as interactions among them. The tool provides the user with the option of selecting multiple views, designed to capture different aspects of the evolving graph from the perspective of a node, a community or a subset of nodes of interest. Standard visual templates and cues are used to highlight critical changes that have occurred during the evolution of the network. A key challenge we address in this work is that of scalability – handling large graphs both in terms of the efficiency of the back-end, and in terms of the efficiency of the visual layout and rendering. Two
case studies based on bibliometric and Wikipedia data are presented to demonstrate the utility of the toolkit for visual knowledge discovery.

6.1 Visualization of Network Content

In Section 5.1, we proposed two types of network content queries: topic modeling and topic event detection. In this section, we describe a visual interface for presenting the results of these queries.

Topic modeling queries find the popular topics from the network content within a query time interval. We approximately reconstruct the network content in the query time interval and call existing topic modeling algorithms to find popular topics. The query results contain two key information of the network content in the query time interval. First, the popular topics are returned and are represented by a set of words. Second, the trending information of each topic can be retrieved from our D-SPUR summary. Our visual interface of the topic modeling query takes into account both of the above information.

Figure 6.1 shows an example of the visualization we designed for topic modeling queries. We use the query results in Table 5.1 as an example. In the right half of the visualization, we gave a list of the topics the same as Table 5.1. This list contains the information about the content of each topic. In the left half of the visualization, we presented a stacked area plot of the trend of each topics detected by the topic modeling algorithm. Recall that the trending information can be retrieved from the time series at the transactions in our D-SPUR summary. From the visualization, the user can get an idea about how these topics change over time.
Figure 6.1: Visualization of topic modeling result.

Topic event detection queries capture the evolution of the topics in two time intervals. We define events such as appearance, disappearance, growth, shrinkage, continuation, merging, split and transformation. The informational results returned by the topic event detection queries include the type of the event, topics before and after the event, and the trending information of the topics in the event. All of these three elements should be presented in our visualization of the topic event detection query results.

To visualize the topics in an event, we can use the same visual interface as Figure 6.1 to present the topics of each time interval separately. We also develop visualization for the events between two time intervals as shown in Figure 6.2. This is a visualization of the events in Table 5.4. We have a tabbed panel to separate different categories of events. In each corresponding tab, we list the events of this category and represent the topics involved in the events with buttons. If the user clicks the button of a topic, the trend of the topic in the corresponding time interval will be
plotted. Figure 6.2 shows the visualization of the merge event in our example. After the user clicks the button of Topic 1 of the first time interval, the trend of the topic in the first time interval is plotted. This visualization not only can present the event detection results to the user, but also can provide trending information of the topics to help the user understand the reason why an event happens.

![Figure 6.2: Visualization of event detection result.](image)

6.2 Visualization of Network Topology and Events

In many scientific domains, visual aids and interactivity are often key to forming important insights, particularly when targeting hard problems. Given the nature of the knowledge discovery process with a human-in-the-loop, a visual analytic interactive front-end is extremely beneficial for effective information synthesis.

In this section, we present such a visual analytic toolkit targeted toward the analysis of large-scale interaction networks. Many real world problems can be modeled as complex interaction networks where nodes represent entities of interest and edges mimic the relationships among them. Fueled by technological advances and inspired
by empirical analysis, the number of such problems and the diversity of domains from which they arise – physics, sociology, technology, biology, chemistry, metabolism and nutrition – is growing steadily. In a large number of such domains the networks governing interactions are known to evolve or change – bibliometric data, social network data, epidemiology data, biological networks, and the World Wide Web to name a few examples.

In such networks, the addition and deletion of edges and nodes can be used to represent changes in the interactions among the modeled entities. The challenge is to identify and localize the portions of the network that are changing to help characterize the type of change and its potential causes, *visually*. A related challenge is to facilitate interactive interrogation, i.e., the user needs to be able to interactively select and zoom down to clusters, entities of interest, as well as specific dynamic interactions and events that govern the evolution of interaction networks over time.

To address these challenges, we introduce a visual toolkit specifically designed to analyze large-scale dynamic graphs. Figure 6.3 provides a schematic representation of the components of our proposed visual analysis toolkit. The back-end of our toolkit leverages a previously developed event-detection algorithm for analyzing dynamic interaction networks introduced in Section 2.1.3. This algorithm presents a methodology to detect critical events affecting nodes and communities in such networks and offers a principled way to characterize their evolution through the composition of various incrementally computable behavioral indices such as stability, sociability and influence. As shown in Figure 6.3, this information is integrated with our front-end. We take the event detection algorithm as an example to show the visualization of
Figure 6.3: Overview of proposed toolkit
graph mining algorithms. This visualization interface can be modified to serve the visualization of other analytical algorithms in Chapter 2.

To facilitate visual analysis, the front-end of the toolkit presents the user with the option of multiple views - a *graph view* which is a cumulative snapshot representation of the graph at different points in time, a *community view* which represents the cluster arrangements of the snapshot graphs, an *event view* which demonstrates the transformations that have occurred over time, and a *node view* which details the evolutionary behavior of individual entities. We allow the user to pick the intervals of interest and drill down onto the corresponding events and behavioral measures within that time-frame. We use a weighting function to associate different behavioral characteristics such as influence and sociability with nodes and importance and recency (temporal stability) with edges. These weights are then mapped onto effective visual cues to localize features of interest. Overall, the front-end conforms to the popular mantra – *overview, zoom, filter and details on demand* [27].

For exploratory visual analysis, timelines of the computation and presentation is important, particularly when considering large real-world graphs such as social networks like Myspace and Flickr. Even simple layout and plotting tools suffer when the size of the graph is very large. For our toolkit, we make use of key optimizations to speed up computation in the back-end, and leverage the use of coarsening mechanisms to provide scalable performance in the front-end to squeeze relevant information in the available pixel space [102].

In short, the challenges that we address in our work are:
1. Identifying, tracking and representing interesting behavioral properties of interactions among nodes and communities such as stability, popularity, frequency etc.

2. Analysis and visualization of communities over time to discover key events depicting changes that occur with respect to other nodes and communities. Also, we target the incorporation of semantic content to rank and evaluate interesting events.

3. Analysis and visualization of the relationships of a node with its neighbors to discover trends in its importance.

4. Ensure scalability to large graphs and facilitate interactive exploratory visual analysis.

We present two case studies on real evolving graph datasets to underline some of the benefits of our toolkit for visual analysis.

### 6.2.1 Datasets

**DBLP Dataset:** We used a subset of the DBLP bibliography ⁹ to generate a co-authorship network representing authors publishing in several important conferences in the field of AI, databases and data mining. We chose all papers over a 10 year period (1997-2006) that appeared in 28 key conferences spanning mainly these three areas. The conferences we considered are - (PKDD, ACL, UAI, NIPS, KR, KDD, ICML, ICCV, IJCAI, CVPR, AAAI, ER, COOPI, SSDBM, DOOD, SSD, FODO, ⁹http://www.informatik.uni-trier.de/~ley/db/
DASFAA, DEXA, ICDM, IDEAS, CIKM, EDBT, ICDT, ICDE, VLDB, PODS, SIGMOD). We converted this data into a co-authorship graph, where each author is represented as a node and an edge between two authors corresponds to a joint publication by these two authors. We chose the snapshot interval to be a year, resulting in 10 consecutive snapshot graphs, containing 23136 nodes and 54989 edges. These graphs are then clustered separately and analyzed to identify critical events and patterns. It has been shown that collaboration networks display many of the structural features of social networks[60, 76]. Hence, this is a good representative dataset for this study.

**Wikipedia Dataset:** The Wikipedia online encyclopedia is a large collection of webpages providing comprehensive information concerning various topics. The dataset we employ represents the Wikipedia revision history and was obtained from Berberich [11]. It consists of a set of webpages as well as links among them. It comprises of the editing history from January 2001 to December 2005. The temporal information for the creation and deletion of nodes (pages) and edges (links) are also provided. We chose a large subset of the provided dataset, consisting of 779005 nodes (webpages) and 32.5 M edges. We constructed snapshots of 3 month intervals, and considered the first 6 snapshots for our analysis.

### 6.2.2 Optimizations for Fast Event Detection

The event detection algorithm (Section 2.1.3) proceeds in an iterative manner, with every two successive snapshots analyzed to compute events among them. So, at each stage, we analyze the respective clusters of $T_i$ and $T_{i+1}$ and compute events between them. First, it is important to note that, since we will be considering only
a pair of timestamps at a time, we do not need to consider all $N$ nodes, since many of the nodes may not be active over the time period. Hence, for event detection between $T_i$ and $T_{i+1}$, we need to examine only the nodes active over either of the two timestamps. This greatly reduces the complexity of the event detection algorithm. Table 6.2 gives the percentage of active nodes, for both datasets. It can be observed that the percentage of active nodes for a pair of snapshots never increases beyond 40% of the total number of nodes.

To facilitate exploratory visual analysis, we need to ensure that event detection can be performed quickly, as the events need to be shown to the user for further analysis. Our detection algorithm relies on finding intersections and unions of the cluster sets, as evident from the formulae presented in the previous section. When the number of clusters is large, finding these intersections and unions can be expensive even with the bit matrix operations we described in our earlier work[5]. Finding the intersection between $k_i$ clusters of $T_i$ and $k_{i+1}$ clusters of $T_{i+1}$ has time complexity $O(k_i \times k_{i+1})$; For most real-world graphs, the number of communities can be quite large ($k_i \times k_{i+1} > N$). The number of clusters obtained for each timestamp of the DBLP and Wikipedia graphs are shown in Table 6.1.

To enhance the performance of the back-end, particularly when scaling to datasets like the Wikipedia data, we develop an optimization to calculate the cluster intersection matrix $I$ in $O(M)$ time, where $M$ is the number of nodes active in either $T_i$ or $T_{i+1}$ ($M \leq N$). The idea is as follows. We first construct two cluster vectors (for the two timestamps considered), to represent the clusters (community) that a node belongs to in a timestamp. We then traverse these vectors sequentially and update the cluster intersection matrix $I$, as shown in Algorithm 1.
Table 6.1: Number of clusters.

![Table](image)

**Algorithm 7** Intersection\((C_i, C_{i+1})\)

**Input:** Set of \(M\) active nodes

```plaintext
for m = 1 to M do
    \(cluster_i[m]\) = cluster id that node \(m\) belongs to in timestamp \(T_i\)
    \(cluster_{i+1}[m]\) = cluster id that node \(m\) belongs to in timestamp \(T_{i+1}\)
end for

// We then traverse these cluster vectors from left to right.
for m = 1 to M do
    if \(m\) is active in \(T_i\) and \(T_{i+1}\) then
        \(I[cluster_i[m]][cluster_{i+1}[m]]\) + +;
    end if
end for
```
The cluster unions can be computed easily by taking the sum of the cluster sizes and subtracting the intersection obtained from $I$.

Note that all the behavioral measures described above can be computed incrementally. We maintain a vector in memory for each of the behavioral indices. As the increments are computed for each timestamp, the corresponding values are updated. Thus, at any given time point, one can obtain the Index values in a straightforward manner. These measures are displayed to the user as part of the node view, which will be described in the next section.

The timing results for the event detection and index computation are given in Table 6.2. To emphasize the savings, we also present the performance of our earlier implementation[5] on the Wikipedia dataset, without the above mentioned optimizations (see Table 6.3). In a nutshell the optimizations are very effective and ensure that the back-end is significantly faster than before and is within very reasonable limits given the scale of the data being operated on.

<table>
<thead>
<tr>
<th>Time stamps</th>
<th>DBLP</th>
<th></th>
<th>Wikipedia</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Active Nodes</td>
<td>Time (secs)</td>
<td>Active Nodes</td>
<td>Time (secs)</td>
</tr>
<tr>
<td>1-2</td>
<td>0.23</td>
<td>0.088</td>
<td>0.03</td>
<td>0.12</td>
</tr>
<tr>
<td>2-3</td>
<td>0.25</td>
<td>0.094</td>
<td>0.07</td>
<td>0.5</td>
</tr>
<tr>
<td>3-4</td>
<td>0.24</td>
<td>0.087</td>
<td>0.13</td>
<td>1.7</td>
</tr>
<tr>
<td>4-5</td>
<td>0.26</td>
<td>0.099</td>
<td>0.19</td>
<td>4.5</td>
</tr>
<tr>
<td>5-6</td>
<td>0.27</td>
<td>0.091</td>
<td>0.22</td>
<td>11.15</td>
</tr>
<tr>
<td>6-7</td>
<td>0.29</td>
<td>0.096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-8</td>
<td>0.34</td>
<td>0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-9</td>
<td>0.41</td>
<td>0.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9-10</td>
<td>0.40</td>
<td>0.14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Computation Times for the Back End.
<table>
<thead>
<tr>
<th>Time stamps</th>
<th>Old[5] (secs)</th>
<th>Optimized (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>10.25</td>
<td>0.12</td>
</tr>
<tr>
<td>2-3</td>
<td>90.92</td>
<td>0.5</td>
</tr>
<tr>
<td>3-4</td>
<td>704.93</td>
<td>1.7</td>
</tr>
<tr>
<td>4-5</td>
<td>2256.34</td>
<td>4.5</td>
</tr>
<tr>
<td>5-6</td>
<td>5016.52</td>
<td>11.15</td>
</tr>
</tbody>
</table>

Table 6.3: Computation Time Comparison.

6.2.3 Visual Analysis

In this section, we highlight the key components of the interface along with associated user interaction features. We also motivate the benefits of these components with respect to the overall goal of knowledge extraction from evolving graphs. The key components of the toolkit are:

**Data Loader:** This component is used for reading the input data and label files. The data to be read is in the form of temporal snapshot graphs, as we described in Section 3. Each graph corresponds to one time step and is stored in an *edge file* format. Additionally, a label file is read which associates each node in the graph with a unique identifier and name, if available. If clusters are already available, then we provide an option to read in the cluster file as well. If not, clustering can be performed online. We provide options for kMetis or MCL clustering. Once the data is read, pre-processing is done to create the *cluster vectors* (described in the previous section) for the first two timestamps.
**View Mode Selector:** Once the data is ready, the user selects one of the four supported views and the relevant parts of the interface get activated. Before detailing the views, we describe our hierarchical representation.

Figure 6.4: Illustration of coarsened high-level view.

**Coarsening:** To visualize large graphs on the screen, we choose to coarsen the graphs, using the cluster information to construct *multi-level hierarchies of nodes* [102]. It provides the user the ability to identify and drill down to sections of interest in the graph. This technique has been validated by Shneiderman et al through user studies. The graph is initially clustered to produce *base clusters* 10. These clusters are then further clustered internally by our coarsening algorithm into multiple levels. The

10If cluster file is already present, we use it
kMetis algorithm is used for performing clustering. Each level consists of a graph of supernodes, each of which represents a cluster of lower-level nodes. At the lowest level in the hierarchy, we have the nodes and edges of the graph. Before coarsening, a new edge file is created by transferring edges between nodes of different clusters to the corresponding supernodes, representing these clusters. This edge file is used by kMetis to obtain the higher level supernodes. The user is initially provided with a high level view in the form of connected supernodes, representing different regions, as shown in Fig 6.4. The physical sizes of the supernodes in the interface reflect the sizes of the cluster they represent. Clusters that contain a large number of nodes can thus be differentiated from singleton clusters with ease. Dynamic behavioral information about the nodes and clusters are also provided, as we will describe below. At any level, the user can select one or a group of interesting supernodes to drill down and visualize the corresponding section of the original graph.

**Graph View**– In this view, the entire dynamic network is displayed as a graph. As mentioned above, the graph is presented as a multi-level hierarchy. The bottom-most level represents the graph itself in the form of nodes and edges. The level immediately above represents supernodes, where each node is a cluster of the lowest-level nodes. Each supernode in this level is labeled with the *Popularity Index* value of the cluster it corresponds to. It is also color-coded to reflect the strength of the *Popularity Index* values. The user can select an interesting set of clusters, and descend to the lower level to visualize the nodes in question. In our implementation, the sequence of colors for nodes (from low weight to high weight) is *Dark Yellow, Light Yellow, Light Green, Dark Green*. Similarly, for edges the sequence is *Dark Red, Light Red, Light Blue,*
The progression of colors for nodes and edges is shown in the bottom right corner of the visual interface. At the lowest level, properties of nodes—sociability, stability, and influence—are computed as described in Section 2.1.3 to assign a weight. Finally, the weights are normalized between [0,1] and are mapped to a color which is then used to render the graphs. We also provide a facility for multi-weighting a node, where we compute the weight taking into account two of these behavioral measures. This is beneficial for discovering correlations among properties of nodes. The relative importance of each edge is primarily captured by its temporal stability, i.e., for how many consecutive time steps that particular edge is observed. Note that, the importance of an edge (interaction) in terms of these measures can be determined based on the nodes involved. For instance, the stability of the edge can be represented as the product of the stability indices of the two nodes i.e \( SI(x, y, T) = SI(x, T) \times SI(y, T) \).

Moreover, to give less importance to old edges (which are not observed recently), we use different line styles. For example, if an edge also occurred in the previous time stamp, we use a dashed line to represent temporal stability. Edges that were not observed recently are represented by a straight line.

**Community View**—This view displays various clusters or communities present in the network. Once the user selects this view, the system presents the user with the clusters that the nodes belong to. The membership of nodes to the clusters are taken into account by using the same color and same marker for rendering.

**Event View**—The event view is designed to provide information regarding transformations that occur in the graph over time. This view displays a set of all critical
events that occur between the current and previous intervals. The user can choose different time intervals and observe the events that transpire among them. Figure 6.5 shows the set of events between clusters of timestamps 2 and 3. At the top of the GUI, there are three bars $a$, $b$ and $k$ which correspond to the $\alpha$, $\beta$ and $\kappa$ parameters for the event detection algorithm described earlier. The user can vary these parameters and examine the critical events obtained. In the middle of the screen, the GUI provides a list of all critical events observed. The user can select one of these events, which provides details on the nodes and clusters involved. We present an example of a Merge and a Split event in the Case Study section. The detailed representation of the event is visualized on the screen giving the user a representation of the nodes involved and the change that has occurred. For the Merge and Split events in the Event View,
we also provide a *Semantic Similarity* ranking. This is of use for graphs that have associated category or term hierarchy information. To begin with, the Information Content (IC) of a term (category or keyword-set), using Resnik’s definition [94], is given as:

\[ IC(k_i) = -\ln \frac{F(k_i)}{F(root)} \]

where \( k_i \) represents a term and \( F(k_i) \) is the frequency of encountering that particular term over all the entire corpus. Here, \( F(root) \) is the frequency of the root term of the hierarchy. Note that frequency count of a term includes the frequency counts of all subsumed terms in an is-a hierarchy. Also note that terms with smaller frequency counts will therefore have higher information content values (i.e. more informative).

Using the above definition, the Semantic Similarity (SS) between two terms (categories) can be computed as follows:

\[ SS(k_i, k_j) = IC(lcs(k_i, k_j)) \]

where \( lcs(k_i, k_j) \) refers to the lowest common subsumer of terms \( k_i \) and \( k_j \). To define the semantic similarity between two clusters, one can employ an information theoretic mutual information measure. Given probabilities of terms \( m \) and \( n \) occurring in a cluster as \( p(m) \) and \( p(n) \) respectively, and their co-occurrence probability \( p(mn) \), the Semantic Mutual Information (SMI) between the two clusters \( C^a_i \) and \( C^b_j \) can be given as:

\[ SMI(C^a_i, C^b_j) = \sum_{m=1}^{k^a} \sum_{n=1}^{k^b} SS(m, n) * \log_{k^a * k^b} \frac{p(mn)}{p(m) * p(n)} \]

However, while this measure accurately captures similarities, it is not very scalable for graphs with large category hierarchies, due to the amount of computation required and memory consumed. In these cases, the semantic similarity between two clusters
can be computed as:

$$\text{Inter}_\text{SS}(C_i^a, C_i^b) = \frac{\sum_{k^a=1}^{m} \sum_{k^b=1}^{n} SS(m, n)}{k^a * k^b}$$

Note that, the semantic similarity values between terms are pre-computed, while computing the Inter\_SS() of clusters or local neighborhoods is scalable. \(^{11}\) Clusters with high values of Inter\_SS(), can be expected to contain authors or webpages with similar topics and thus merge events that comprise of such clusters are semantically meaningful (Semantic Merges). Hence, the Merge events are ranked in decreasing order of the Inter\_SS() of the merging clusters. For the Split events, we compute the Inter\_SS() of the split clusters. We will illustrate both types of Semantic events in the next section. Note that our toolkit can output Semantic Similarity scores for clusters (not shown).

**Node View**— All the above-mentioned views deal with global properties of the network. The node view presents the user with localized information. Once the user chooses node view, the toolkit provides a list of nodes ranked in decreasing order of the properties available (sociability, stability and influence). The user can then select a node from the node list for further observation. This prompts the corresponding neighborhood graphs of that particular node over time to be displayed to the user. The displayed graphs includes the chosen node and its neighborhoods over time. One can gain insight into changes occurring in the neighborhoods of the selected node. For instance, in the case of influence, one can identify spheres of influence for a node over time. We will demonstrate the benefits of the node view in the case studies in the next section.

\(^{11}\)Also, note that these operations are performed only on merge and split events detected.
**Zoom Filters:** As the name suggests, this feature is used to zoom into certain sections of the graph. The user can select the area of interest by drawing a rectangle using the mouse. The selected part is then zoomed into and displayed. It is also possible to zoom out to a lower resolution. Figure 6.6(a)-(b) demonstrates the zoom feature on the DBLP dataset. The zoom feature can be used multiple times to increase the resolution.

![Figure 6.6: Zoom feature](image)

**Time Browser:** This functionality is used to observe the network across time. This provides the user the capability to detect time instants when graph topology has changed considerably. The *Back* and *Forward* buttons at the bottom of the GUI can be used by the user to control the time, moving through the different time intervals.

**Keyword Search:** In many cases, one is interested in searching for particular nodes [55] and their evolution over time. As we described above, we use a multi-level hierarchy of supernodes to visualize sections of the graph. We also store a
multi-level index that allows the querying and extraction of specific nodes of interest. If one is interested in a particular node and its behavior, we can extract the exact location of the node and visualize the corresponding section of the graph.

**Evolution of Neighborhoods:** We also provide the option of visualizing the evolution of node neighborhoods. When the user is presented with a view of a region of the graph, she has the option of selecting a particular set of nodes or communities and viewing the evolution of the neighborhoods of these nodes over time. Once a region is chosen, we track the neighborhoods of all nodes in that region over time.

**Community-driven Layout:** An integral component of our visual framework is the layout component. Once we have a set of nodes and edges to render on the screen, we need to map them to suitable coordinates that represent the relationships that exist among them. As a first step we leveraged the Graphviz(http://www.graphviz.org/) layout tool to obtain coordinates. However, since the graph is dynamically evolving, we observed that directly employing Graphviz led to a loss in visual correspondence of nodes and more importantly communities across timesteps. Resolving this problem is not straightforward since nodes and edges may not be active at all time points. To handle this correspondence problem, we compute a novel *community-driven* layout scheme. The central idea is to ensure that communities that overlap across time steps are laid out in corresponding regions in consecutive time steps, thus maintaining relatively stable coordinates over time. The heuristic procedure we adopt involves identifying the most stable communities (using the *Continue* and *Stability* computations
described earlier) and suitably ensuring that global coordinates for such communities from their formation until they change drastically, remain consistent. This leads to the desirable property of visual correspondence across time stamps. Note however that individual nodes that join or leave communities are moved around based on their behavior, as are communities that change significantly.

6.2.4 Case Studies

DBLP Bibliography Dataset

Figure 6.7: Node View: Neighborhood of R. Agrawal

In this case study, we demonstrate the effectiveness of our toolkit for visual data analysis on the DBLP dataset. Our tool provides us with a list of authors ranked by
behavioral attributes such as Sociability and Influence, as described previously. Our tool also allows one to combine information from multiple metrics by specifying an affine combination of these values (menu-driven option not shown). For this study, we chose Dr Rakesh Agrawal, who unsurprisingly ranks highly on both sociability and influence (see Fig 6.7). We equally weighted the contribution of each index.

Upon inspection, one can see that the neighborhoods for Dr Agrawal differ significantly between successive snapshots. From 1997 to 2002, one can make out the progression in his sociability and influence index, as conveyed by the gradation of the color of the node representing him. After 2002, however, many of his neighbors (collaborators) remain fairly consistent, the sociability index is lower but the influence, of collecting more neighbors in the cluster he is in balances this out very nicely. This correlates with his interests shifting to the focused area of privacy preserving data mining and trust and security applications of databases. His collaborators in the last few timestamps shown are primarily from this area and this area has also taken off very nicely since Agrawal and Srikant’s seminal paper in the area in 2001.

Also, one can readily see that 3 nodes in particular appear quite frequently, namely R. Srikant, R. Bayardo and J Kiernan (after 2001) represented by dashed edges. These are some of Agrawal’s frequent collaborators. Another interesting trend is that the graph also identifies quite naturally collaborators of Agrawal’s who have a high sociability and influence index (i.e. Christos Faloutsos, Gerhard Weikum, Dimitrios Gunopulos, Johannes Gehrke, Prabhakar Raghavan, and Surajit Chaudhary).
In the next case study, we analyze the Wikipedia webgraph. In particular, we demonstrate the use of Semantic analysis with event detection. In the event view, the toolkit provides us with a list of different events. As we mentioned in the previous section, we have the facility of ranking the Merge and Split events in terms of semantic meaningfulness. First, we will consider a *Semantic Merge* event, that had the highest Semantic Similarity, shown in in Figure 6.8(a). In the first snapshot, there are 2 clusters of size 5 and 11, that have considerable semantic similarity, as can be ascertained from the labels. The clusters deal with *Logic* and *Philosophy* and the Merge event is thus justifiably meaningful. The merged cluster is shown on the right.

For a Split event, one would expect the two split clusters to be semantically dissimilar. While, this is mostly true, there can be occurrences where interesting minute

\footnote{In 1997 this value is low simply because this is the first data point in the dataset we use – an artifact of the experiment.}
differences can cause clusters to split up. These kind of Split events can indicate subtle changes across snapshots, where a cluster splits into two parts due to a small semantic difference among the associated categories. These splits can be interesting as they can reveal differences that may not be obvious. This can be considered akin to drilling down a hierarchy to discover subtle specializations of a category. We can find such interesting occurrences by considering split clusters with low Inter\_SS(). An example of such a Split event is shown in Figure 6.8(b). As can be observed, the original cluster which deals with the Berlin Wall, splits into two clusters on East and West Berlin respectively. Understandably, these two split clusters had reasonably high Semantic Similarity values.

6.3 Conclusions

We have presented a toolkit for visualizing and analyzing dynamic interaction graphs. Our toolkit provides multiple views of the data and is designed to incorporate features for multi-scale and multi-resolution analysis and supports the overview, zoom, filter and details-on-demand paradigm.

We have shown how the toolkit can be employed for estimating interesting behavioral properties of nodes and communities in the graph, such as stability and influence. The node view also allows one to visualize trends in these properties over time. Additionally, the graph and event views permit us to discover and characterize specific changes that occur, with respect to other nodes and communities. We have shown how the incorporation of semantic content can aid evaluation and ranking
of events discovered. Our novel community-driven layout component can aid exploratory analysis and handle the correspondence problem in plotting trends of nodes and communities over time.

To ensure scalability to large graphs, we have presented optimizations that can provide speedup and facilitate interactive visual analysis. The toolkit also supports visualizing the cumulative graphs with different scoring mechanisms to assign color to each edge and node. The coloring scheme captures behavioral properties and provides useful visual cues to discover important and interesting parts of the graph. We have shown how the toolkit can perform visual analysis by taking into account the evolution of nodes and communities as well as key events over time. Using illustrations on the DBLP and Wikipedia datasets, we have shown how the interactive features aid the user in answering common queries about dynamic networks in an effective and efficient manner.
Chapter 7: Conclusions and Future Directions

In this dissertation, we proposed a framework to speed up analytical problems on large-scale network data and reduce the storage size of the data. Our framework can also support novel query mechanisms and effective visualization to enable the discovery of new knowledge. Our framework consists of four main stages: a high performance computing platform to speed up mining algorithms, a data summarization step to reduce the memory size of the data, a complex query engine to discover important knowledge and patterns, and a visual front-end to present the mining results to end-users. In Chapter 3, we discussed how to leverage modern parallel GPU computing architectures to speed up a series of widely used graph mining algorithms. In Chapter 4, we proposed a pattern-driven data summarization approach to reduce the size of the streaming network content and network topology. The summary we built can be directly used by mining algorithms. We proposed complex queries of the summarized network data in Chapter 5. Finally, we developed a visual front-end to present the mining results in Chapter 6.
7.1 Contributions

Next, we give a brief description of the key contributions made in this dissertation and the limitations of our solutions. This thesis has focused on improving the scalability of analyzing large-scale networks, developing new analytical queries to discover new patterns and visually presenting the analytical results.

7.1.1 High performance mining kernels

We studied the common properties and performance bottleneck of graph mining algorithms including PageRank, HITS and Random Walk with Restart. We identified a computational kernel shared by these algorithms, the sparse matrix-vector multiplication (SpMV) kernel. We developed optimizations on the GPU architectures to speed up this kernel and thus improve the performance of the investigated graph mining algorithms. Using real dataset, our optimizations can yield about 2x speedup over competing GPU methods and about 30x speedup over CPU methods. We also extend these optimizations to a cluster environment with multiple GPUs. This extension enables our high performance computing platform to handle out-of-core datasets which cannot fit in the memory of a single GPU. Moreover, the high performance of our optimized kernel relies on significant programmer expertise to tune parameters of the approach. To alleviate this, we developed a systematic mechanism for tuning the parameters automatically at runtime depending on the characteristics of the input data.
7.1.2 Summarization of network content and topology

Besides speeding up the mining algorithms, we also proposed novel approaches to improve the scalability of large-scale network analytics in an orthogonal direction by reducing the size of the input data with data summarization techniques. Our data summarization methods can reduce the storage size of the streaming network data which contains both network content and topology. This approach can alleviate the pressure on the main memory of data processing systems posed by the large-scale input data. The compact summary can not only reduce the amount of memory required to store the data, but also can be directly used by mining algorithms without decompression and can speed up these mining algorithms. Experimental results show that our method can efficiently and incrementally summarize the network content and topology. When summarizing the network content as a high speed message stream, the memory footprint of the summarization algorithm grows logarithmically with time instead of linearly. The raw data can be approximately reconstructed by querying the summary so as to support the analytical applications over the original data. When summarizing the network topology, our lossless compression method can yield up to 2x compression ratio on the graph representing large social networks and the Web. The summary can be directly used in the high performance sparse matrix and vector multiplication kernels, and lead to corresponding speed-ups in the performance of graph mining algorithms.

7.1.3 Complex query of network content and topology

On top of the high performance computing platform and the compact storage of input data, we proposed novel analytical queries to facilitate the discovery of new
knowledge from modern social networks such as Twitter. For analyzing network content, we propose topic event detection queries to detect the evolution of the popular topics. For network topology, we project the entire network topology onto a content-driven temporal subgraph. Then graph mining algorithms such as link analysis, graph clustering and event detection can be performed on the subgraphs. We show examples of query results from the Twitter social network data.

7.1.4 Interactive visual front-end

We developed a visual front-end to interactively present the query results to end-users. In the visual-analytic toolkit, we integrated various visual features to display the mining results from different aspects. The toolkit is designed with multi-level visualization of the graph data and supports the overview, zoom, filter and details-on-demand paradigm. To ensure scalability to large graphs, we have presented optimizations that can facilitate interactive visual analysis. We have shown how the interactive features can aid the user in understanding information about static and dynamic networks in an effective and efficient manner. The interaction with the end-users can form a feedback loop to the mining process, thus the users with domain knowledge can guide the system to find more meaningful patterns.

7.2 Limitations

In this dissertation, we propose a framework to support the large-scale analytics of network data. However, there are some limitations in our solutions and evaluations.

With respect to the work presented in Chapter 3, where we demonstrate the use of GPUs to speed up a series of graph mining algorithms. A limiting factor on the effectiveness of this approach is current PCI-express technology. The PCI-Express
bus that connects the CPU and GPU is a limiting factor due to its low memory bandwidth. To handle the computation of large-scale mining problems, we used multiple GPUs to store the web-scale graph data and avoid data transfer overhead between CPU and GPU. New computer architectures such as Intel Sandybridge and AMD Fusion combine CPU and GPU together on one chip. The PCI-Express bus problem is unlikely to be a limitation in such architectures while our optimizations can still be applied on the GPUs of such chips.

With respect to the work presented in Chapter 4, in our summarization and query processing component, we use a pyramidal time window to maintain a stream of summary objects in our D-SPUR algorithm. This process is lossy with higher accuracy in recent and frequent data than historical and infrequent data. This is because the D-SPUR algorithm is based on the assumption that users are more likely to query recent and frequent data. We use random sampled keywords as queries to evaluate the quality of the summary built by our D-SPUR algorithm. Also the summary generated by our G-SPUR algorithm to accelerate the computation of PageRank on large graphs has the following limitations. Our method requires that the input graph is unweighted. The acceleration comes from the computations saved when the virtual nodes (or patterns) first aggregate the rank of all nodes in it, and then push the total rank evenly to their shared out links. An arbitrarily weighted graph does not meet the requirement of the uniform out-link distribution.

With respect to the work presented in Chapter 6, the primary limitation of our empirical study is that we have not conducted a full blown of user study to evaluate the effectiveness of the front-end of the visual interface. In this work, we primarily focused on the efficiency of the back-end of the visual analytic framework. However,
we would like to point out that our front-end interface follows the commonly accepted paradigms such as overview, zoom, filter and details-on-demand. These techniques have been thoroughly evaluated by the information visualization communities [27].

7.3 Future Directions

Next, we will discuss some possible extensions and future directions.

7.3.1 Cloud based large-scale graph mining

In this dissertation, we used a high performance computing platform with GPUs interconnected in a cluster environment to speed up large-scale graph mining problems. Cloud computing platforms based on the Map-Reduce programming paradigm has been widely used in large-scale data processing. The advantages of these systems include high reliability, scalability and productivity. There has been effort to implement graph mining problems on the cloud based systems such as Pegasus [56] and Pregel [72]. Jiang and Agrawal [51] proposed an alternative API of Map-Reduce on HPC platforms, which can explicitly maintain reduction objects and speed up graph mining applications. An ideal system for large-scale graph mining should be provided with the reliable, scalable and productive advantages of Map-Reduce systems as well as the high absolute performance of HPC platforms. To achieve this goal, we think the following research problems need to be solved.

• Partition and placement of large graphs in a distributed system. The partition of a large graph on different computational nodes decides the communication overhead in the execution of the graph mining algorithms. Thus, the placement of the graph data should be optimized to minimize the communication cost.
• Low-level cache and memory locality. While emphasizing the productivity of high-level programming APIs like Map-Reduce, it is crucial to improve the absolute performance of a data processing system by optimizing low-level memory access localities. Such optimizations are desirable to be transparent to graph mining application developers due to productivity and reliability considerations.

7.3.2 Applications of data summarization on mobile devices

Mobile devices like smart phones and tablets are becoming more and more popular. With the powerful CPU and fast internet connections, a large amount of data is processed and generated on these devices. However, the limitation of memory and storage space is inevitable because the compact size and light weight of these devices. This limitation presents a great opportunity for our data summarization method to reduce the storage size and only keep the important data in the limited memory of mobile devices. Data summarization techniques can also alleviate the burdens on the network bandwidth and the battery life of the mobile devices. But it is challenging to define a generalized data format that can be used by most of the applications on these mobile devices and the generalized data should be efficiently summarized on the smart phones or tablets with very low computational and storage overhead.

7.3.3 Management and complex query of graph database

It is desirable to integrate the components in our proposed framework into a database system. The end-users only need to query the graph database with a well defined query grammar. The system should be able to automatically handle data management and computational tasks such as summarizing input data, computing graph mining kernels and translating the queries. Such a system can make broader
impact in real-world applications by using the method proposed in this dissertation. However, there are several challenging research issues in building such a system. For instance, we need to define the query grammar according to the analytical tasks we want to support on the large-scale network datasets. We also need to build a compiler middleware to translate the user-input queries into computational tasks in our framework and to schedule these tasks.

7.3.4 Improvement on user interactions

The interactions between data mining systems and end-users are always important because the human-in-the-loop property in the knowledge discovery process. The studies of this field lie in the intersection of computer science, behavioral science, design and etc. In large-scale data mining, it is even more challenging due to the unlimited number of data elements. In this dissertation, we presented our initial efforts of visualizing the results of large-scale network analysis. In terms of visualization, our work can be improved with new visual features such as temporal visualization of network content, better coarsening of large network topology by graph sparsification and clustering and so on. Besides visualization, there are also other possible options worth investigating to improve human and computer interaction for data mining systems. For example, the design and automated recommendation of queries can guide users to discover more valuable information from a large-scale dataset.
Bibliography


[29] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *KDD ’07:*


[39] Peter A. Gloor, Rob Laubacher, Scott B. C. Dynes, and Yan Zhao. Visualization of communication patterns in collaborative innovation networks - analysis of some w3c working groups. In Proceedings of the twelfth international conference


[81] NVIDIA. *Optimizing Matrix Transpose in CUDA*.


[117] Dong Xin, Jiawei Han, Xifeng Yan, and Hong Cheng. Mining compressed frequent-pattern sets. In *Proceedings of the 31st international conference on Very large data bases*, VLDB ’05, pages 709–720. VLDB Endowment, 2005.


